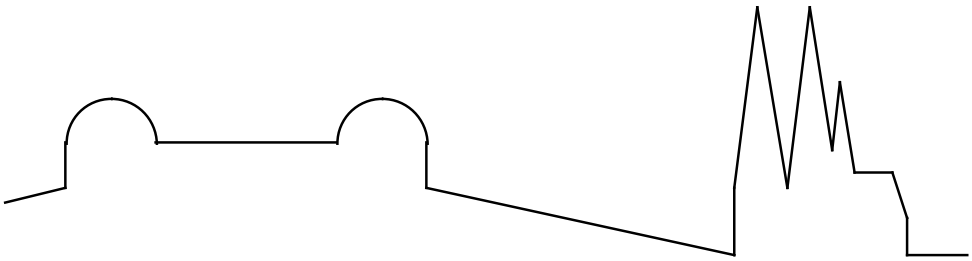
PAQS

Data Sources, Exchange Formats, and Database Schemas for a Proteo-Chemometric Analysis and Query System

Fredrik Bökman



Thesis for the Degree of Master of Science
Majoring in Computer Science, 20 credit points
No. nn/2001

ISSN 1100-0686

Department of Information Science
Computer Science Division
Uppsala University
P.O. Box 513
S-751 20 UPPSALA

Sweden

Abstract

The research described in this Thesis is part of a project to develop a new database system for proteo-chemometric research. This new system uses a mediator/wrapper approach for integrating heterogeneous and autonomous data sources. Special-purpose modules for data representation and data analysis can be incorporated into the system through the extensibility of the object-relational mediator.

Life science data sources and data exchange formats for the new Proteo-chemometric Analysis and Query System (PAQS) have been surveyed. Although important data sources exist on many different formats the trend towards XML is evident. For proteo-chemometric research it is important to be able to access data sources with binding affinity data. Most such data sources are only accessible via web forms, which limits the query capabilities.

Database schemas for parts of the proteo-chemometric information domain have been developed within a functional data model with object-oriented extensions. These schemas have also been implemented in the Amos II system as a first-stage prototype of PAQS. Special emphasis has been put on modelling binding experiments and experiment evaluations, and the corresponding data types have been used to show how data analysis could be performed by means of foreign functions of the mediator.

The mediator/wrapper approach is described in the Thesis, and examples are given of other systems which use this architecture for integrating life science data, both research prototypes and commercial systems. Introductions to the proteo-chemometric approach to drug design, to some general database concepts, and to information integration by means of database systems are also given.

Contents

Supplementary electronic material:

"Paqs_files.zip" contains type definitions for the prototype plus the following files:

"Paqs_prot.amosql" (script for loading the proteo-chemometric data types), "BindAid-_appendix.amosql" (loads the prototype database with data from appendices 4 and 5 of the BindAid manual), "Example_queries.amosql", "ChemicalData.mbd", "Register_ODBC_data-_source.txt", "External_demo.amosql", and "External_IUT_demo.amosql".

The supplementary material is available as a whole or as separate files from the author upon request (email: fbn@hig.se). The Amos II system must be obtained separately, and can be downloaded free of charge (after registration) from http://www.dis.uu.se/~udbl/amos/ (2002-01-16).

1 Introduction

The research described in this Thesis is motivated by the need for new approaches to the development of drugs. Generally speaking, there are two steps in drug development: First a key molecule, usually a protein, is identified. This "target protein" can have some biochemical function which, e.g., causes a disease. Then a drug molecule which moderates, or blocks, the function of the target protein is searched for. The proteo-chemometric approach to drug design (chapter 2) is one of many approaches to performing this second step efficiently.

The work of this Thesis is part of a joint project between pharmacologists and computer scientists in order to develop a Proteo-chemometric Analysis and Query System (PAQS). More specifically, the PAQS project emanates from a need for a database system suitable for proteo-chemometric research. The aim of this new system is to provide users with uniform access to world-wide data on bindings between target proteins and potential drug molecules, as well as associated data. Further, PAQS shall provide the users with computational tools to analyse such ligand-protein interactions - together with additional data - in order to produce new knowledge. In order to accomplish this it is necessary for the new proteo-chemometric analysis and query system to integrate data from various autonomous and heterogeneous life science data sources.

Four of the first steps towards the development of PAQS are (i) to develop an architecture for the system, (ii) to survey potential data sources and trends in life science data integration, (iii) to develop database schemas which represent the problem domain, and (iv) to develop methods for importing data from external sources. This Thesis deals with points (ii) and (iii). The general mediator/wrapper-architecture (step (i)) is described in section 7.1. With this architecture external data are imported through wrappers (step iv).

1.1 Parts and Purposes of the Thesis

This Thesis consists of three main parts. First, introductions to proteo-chemometric research, to database concepts, and to information integration are given in chapters 2, 3, and 4, respectively. These chapters are meant to provide the reader with a basis for the later parts of the Thesis.

The second part consists of chapters 5 and 6. Chapter 5 surveys life science data on the web, focussing on the usefulness for the PAQS project. The chapter discusses data formats and protocols, as well as specific data sources. Chapter 6 surveys some work related to the PAQS project, viz. database solutions for the integration of heterogeneous life science data. One purpose of this part of the Thesis is to show solutions that have been used previously for standardisation and integration in this information domain. Further, the chapters should present current trends in order to give a good basis for the selection of data formats and data sources to be used in the PAQS project. Finally, the chapters should point at solutions in database and information system design which can be reused in PAQS.

Chapter 8 constitutes the third part of the Thesis. In this part database schemas for radioligand binding experiments, evaluations of experiments, and related topics are described and discussed. The purpose of the chapter is to suggest design solutions for most subdomains of the PAQS information domain. The proposed database schemas should also be useful as templates for future XML schemas of binding experiments and similar topics. A third purpose with the chapter is to demonstrate how data analysis can be performed from within the

database manager, by the invocation of foreign functions. The design is supported by a prototype implemented in the Amos II system.

1.2 Tools

In order to prepare the surveys of chapters 5 and 6 primary and secondary literature was studied. Most of this information is available on the Web rather than in refereed printed sources. Since most of the web sites used in the survey were managed by the same organisation that manages the corresponding data source, data format, interchange protocol, et cetera, I regard the web site information as being authoritative, and fairly reliable. Some organisations may perhaps be a bit too positive in their judgements of their own products.

The modelling and implementation described in chapter 8 was performed within the Amos II data model, a functional data model with object-oriented extensions (see section 3.2.6). Several Amos II releases have been used for implementing the evolving schemas. The latest version the prototype has been tested with is Amos II Beta Release 5, v6. A few tests in section 8.8.3 and Appendix F used a developer's (non-public) version (Amos II Beta Release 5, v12, "camos"). Foreign functions were implemented in Java (JDK 1.3). Since the Java programs were small, no interactive development environment was used. Diagrams were drawn with Edge Diagrammer (v4.0) from Pacestar Software.

1.3 Acknowledgements

2 Proteo-Chemometric Research

Proteo-chemometrics[1] is a novel approach to the analysis of drug receptor interactions, which is an important component of drug design. In this Thesis relevant data sources, data formats, and design solutions for PAQS (Proteo-chemometric Analysis and Query System) will be described. The purpose of PAQS is to provide an information system for proteo-chemometric and other pharmacological research. In this chapter a brief description of the background to the proteo-chemometric method is given, without going into details either in biochemistry or statistics. The chapter is intended to give computer scientists and other readers not versed in biology or pharmacology a relevant background for the rest of the Thesis. In the last subsection (section 2.5) a few hot topics - or "buzz-words" - are explained.

2.1 Binding Affinity Constants and the Action of Drugs

Much of the modelling in chapter 8 of this Thesis deals with how to represent binding affinities, experiments made to determine binding affinities, and evaluations of these experiments. Hence, we start by describing why such binding affinities are interesting.

Many drugs contain an active substance which competes for some naturally occurring substrate in binding to an enzyme or to a receptor. *Enzymes* are catalysts in the chemical processes of living systems. A biochemical process which by itself proceeds very slowly may be accelerated by many orders of magnitude (10^6-10^{12} times faster[2]) by an enzyme. Enzymes are also very specific, i.e. a given enzyme catalyses one particular reaction. Now, if an active drug substance binds to the enzyme its catalytic activity might become reduced, or even fully cancelled. Most known enzymes are proteins[3].

A *receptor* is a protein which binds a specific extra-cellular signalling molecule (e.g. a hormone) and then initiates a cell response. Receptors, like enzymes, are very specific, both with respect to which molecules they bind and which cell response they initiate. If an active drug substance binds to the receptor, the receptor becomes blocked, and signal transfer through the cell membrane is inhibited.

Thus, in both cases described above the active drug substance functions by binding to some *binding site*. Obviously, it is essential to know how strongly a substance (usually called *ligand*) binds to various binding sites in order to understand or predict which effects the substance has (or will have) on a biochemical process.

If a ligand binds strongly to a binding site we say that the site has a strong (or high) *affinity* for the ligand. For a quantitative measure of affinity we define the (binding) *affinity constant* (K_A) as

$$L + S \rightleftharpoons LS \qquad K_A = \frac{[LS]}{[L]\,[S]} \qquad \textbf{(Equation 1)}$$

[1] Lapinsh, Prusis, Gutcaits, Lundstedt and Wikberg 2001; Prusis, Muceniece, Andersson, Post, Lundstedt and Wikberg 2001; Prusis 2001.

[2] Lodish, Berk, Zipursky, Matsudaira, Baltimore and Darnell (2000).

[3] "Ribozymes" are RNAs with catalytic activity.

where LS is the complex between ligand and binding site, and square brackets ([ ]) denote concentrations. This definition of affinity constant is the inverse of the dissociation constants used in chemistry.

Traditionally, there are three important ways to determine affinity constants:

- Biochemical experiments with binding assays (see next section).
- Molecular modelling: Computer simulations of differences in free energy between the states {free ligand + free protein} and {ligand bound to protein}.
- QSAR (Quantitative Structure-Activity Relationship): The affinity of a new ligand is predicted from known affinities of similar ligands with help of statistical methods.

In the following sections binding assays and chemometrics (including QSAR) are described. Molecular modelling is not treated in this Thesis, although, in principle, all affinity data would be of interest for the PAQS project.

2.2 Binding Assays

An *assay* is an experimental procedure or environment used to detect and quantify proteins. Each assay relies on some highly distinctive and characteristic property of a protein, e.g. the ability to bind a particular ligand, to catalyse a particular reaction, or to be recognised by a specific antibody[2]. Furthermore, a useful assay must be simple, fast, and sensitive towards the protein[4]. In this Thesis we are mainly interested in *binding assays* performed to determine affinity constants between receptors and ligands.

2.2.1 Radioactive Labelling

As discussed above, knowledge about receptors is essential to the design of new drugs. However, many receptors are difficult to identify and purify since they are present in minute amounts in the cell, and since there are furthermore large amounts of other proteins present[5]. Usually, receptors are detected, and quantitatively measured by their ability to bind radioactive ligands to a cell or cell fragment. The advantage of using radioactive labelling is that there is a very small background signal[6]. Thus, with a radioactively labelled substance in the sample we know that the signal recorded by the detector is due only to this substance, irrespective of what other substances that are present. Obviously, there are disadvantages with radioactive labelling, too. E.g., the labelled substance must be synthesised and it may not have exactly the same biochemical properties as the corresponding unlabelled substance. Furthermore, we introduce a new health hazard when working with radioactivity.

[4] Many common protein assays only require 10^{-9} to 10^{-12} gram of material (Lodish et al 2000, p 90).

[5] Typically, a cell bears 10000-20000 receptors for a particular hormone. This is about one millionth of the total cell protein content (Lodish et al 2000, p 859).

[6] A low background has two advantages: First of all we can collect more of the interesting signal without saturating the detector or signal collection system. Secondly, with a low background, we have a better chance of getting a satisfactory signal-to-noise ratio.

2.2.2 Binding Experiments

In a *competition binding experiment* the assay is made with one (or several) binding sites and one radiolabelled ligand (radioligand) of known concentration. Then a series of measurements are made for varying concentrations of a competing, non-labelled, ligand (the competitor). The concentration of *bound* radioligand is determined by measuring electrons emitted by radioactive decay of the radioligand. This is the only concentration actually measured since the concentrations of the competing ligand are "known", prepared by some simple series of dilutions. As the concentration of competing ligand increases, the concentration of bound radioligand will decrease, see Figure 1a. Thus, in a competition experiment there is one independent variable, the varying competitor concentration, and one dependent variable, the bound radioligand concentration[7].

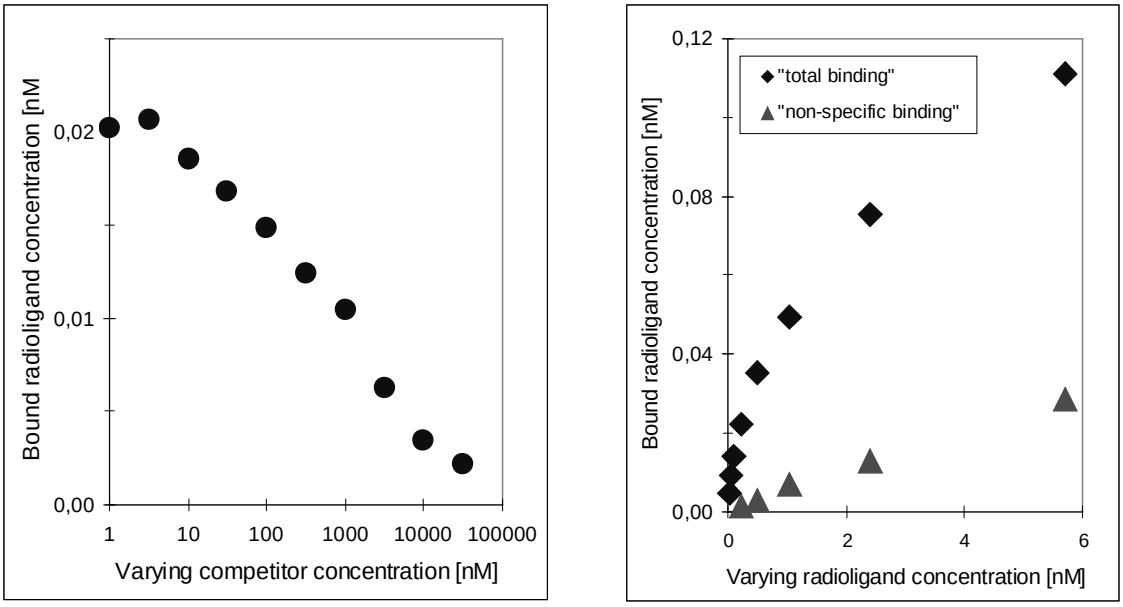


Figure 1. Diagrams of concentration binding experiments examining the binding between the radioligand [^{3}H]MK912 and α_{2A}- and α_{2C}-adrenoreceptors in cerebral cortex membranes. (a, left) Competition curve of guanfacine with 0.22 nM [^{3}H]MK912. (b, right) Saturation curves with (triangles) and without (diamonds) 20000 nM BDF8933. Data taken from the BindAid manual (Wikberg 2001).

In a masked competition experiment there is one (or several) additional non-labelled ligands with *fixed* concentrations, also competing for the binding site. In the database schema developed for the prototype no conceptual difference will be made between masked and non-masked competition experiments. A masked competition experiment is distinguished by having more than one ligand with fixed concentration. (For a competition experiment there is always one: the radioligand.)

In a *saturation binding experiment* it is instead the concentration of the radioligand which is varied. Usually two measurement series are made: the "total binding" of the radioligand is determined in an experiment without competing ligand, and the radioligand's "non-specific

[7] It is also possible to perform a competition experiment with several varying competing ligands. Although this is not usual (if it ever has been done), the database schema of section 8.7 allows for this. However, we will assume that no experiments are made with several radioligands at the same time. As discussed in section 8.2, we will allow several dependent variables, but they origin in parallel experiments, and refer to the same radioligand.

binding" is determined in an experiment with an excess of saturating competing ligand[8], see Figure 1b. Thus, in both these series it is the varying concentration of added radioligand that is the independent variable, and the concentration of bound radioligand that is the dependent variable. The two series differ in that when the non-specific binding is measured there is also a non-labelled competitor of fixed concentration. To get the "specific binding" of the radioligand one finally subtracts the "non-specific binding" from the "total binding".

Both competition and saturation binding experiments study the reversible binding of ligands to receptors (or binding sites). A single such experiment, or a number of them taken together, may be analysed by the model[9]

$$B_i = \sum_{b=1}^{n} \frac{K_{ib} F_i R_b}{1 + \sum_{a=1}^{m} K_{ab} F_a} + N_i F_i \qquad \textbf{(Equation 2)}$$

where B_i and F_i are the bound and free concentrations of ligand *i*, K_{ab} is the binding affinity constant of ligand *a* and site *b*, N_i is the non-specific binding parameter of ligand *i*, and R_b is the concentration of site *b*. Only one of the B_i:s is measured - the bound concentration of the radioligand[10].

Both saturation and competition binding experiments are of the more general type *concentration binding experiment*. There are also other types of binding experiments, e.g. *time binding experiments*. In the latter the rate of dissociation (or association) of complexes between binding sites and ligands are studied. Time is the independent variable, and the bound concentration of radioligand is the dependent variable. The rate of dissociation can be analysed with the following model[9]:

$$B_i(t) = \sum_{b=1}^{n} R_{ib} \, exp(-K_{ib} t) \qquad \textbf{(Equation 3)}$$

Here, *t* is the time, K_{ib} is the rate constant for dissociation of ligand *i* from binding site *b*, and R_{ib} is the concentration of bound ligand *i* to site *b* at time $t = 0$.

In order to get satisfactory statistics it is customary to perform "parallel" experiments: Several measurement series with the same experimental set-up (same assay, same ligands, same varying concentrations) are performed. Parallel experiments will be handled as being several dependent variables in the database schema (section 8.2.3).

2.3 Chemometrics

Chemometrics is ”the use of mathematical and statistical methods for handling, interpreting, and predicting chemical data” (Malinowski 1991). The classical example is that we wish to

[8] I.e., it is assumed that the concentration of the competitor is large enough to completely block out the radioligand from the receptor binding site. Hence, any radioligand bound in the assay is bound "non-specifically".

[9] In the fit program BindAid (Wikberg 2001) these models are called "Bindfit" and "Dissocfit", respectively.

[10] It is assumed that there is only one radioligand in the assay, and hence only one ligand contributing to the detected signal, plus background radiation and noise. Other, non-labelled, ligands could *in principle* be detected by other means - in practise this is very difficult. The schema to be described in this work will, however, accommodate measurements of several bound ligands in a single experiment.

optimise the yield of product C in the chemical reaction A + B $\not\rightarrow$ C + D. Since there are a multitude of parameters we can vary (solvent, catalysts, temperature, pressure, stoichiometric ratio of A and B, rate of addition of B to A, and so on - depending on the kind of reaction) the search space is large, and the parameters may be correlated. Naive optimisation methods will require long series of expensive and tedious experiments, and will generally be stopped when a "satisfactory" experimental set-up has been found. The chemometrical approach would be to first design a set of experiments which span the search space, perform these experiments, and then use multivariate statistical analysis to explore the search space.

Other applications of chemometrics are calibration of instruments, QSAR, and analysis of spectra and chromatograms. Typical multivariate methods are principal component analysis (PCA), factor analysis, discriminant analysis, and projection to latent structures partial least squares analysis (PLS). Sometimes also pattern recognition, robotics and similar methods are counted as chemometrics[11].

In QSAR (Quantitative Structure-Activity Relationship) we wish to take a set of molecules with known experimental data and predict (or explain) some activity with help of the molecular structures. In order to put this quantitatively we obviously need to describe the molecular structures by some variables, in chemometrics called *descriptors*.

Many descriptors are real-valued variables on a linear continuous scale (e.g. mass, logP, and dipole moment), but other types are also possible. For example, descriptors may be discrete variables (number of atoms in the molecule), or binary variables ("presence on non-presence of nitrogen in the molecule"). PAQS should be able to handle all kinds of descriptors, and this requires some careful data modelling in order to get a good design of the database schema.

If we now wish to predict binding affinity constants the approach is as follows: (i) Find as many affinity constants as possible for the receptor of interest, either from the literature or from your own experiments. (ii) Choose those affinity data which seems trustworthy, and where the ligand can help spanning up a reasonable search space. (iii) Construct descriptors for the ligands. (iv) Perform a statistical analysis to build a statistical model. (v) Assess the validity of the model. (vi) Use the model to predict the affinity constant for one or several new ligands. If a high affinity is predicted for a ligand, obviously its actual affinity needs to be determined experimentally, too.

2.4 Proteo-Chemometrics

Proteo-chemometrics is "chemometrics for the analysis of protein-ligand interactions" (Lapinsh et al 2001). The novel thing about the approach is that the descriptors refer to both ligands *and* receptors (or enzymes). I.e., we take affinity constants for a set of similar receptors and a set of ligands, and build the statistical model from all these. This has several advantages: We can find a larger number of relevant affinity data and, more interestingly, we span up a new kind of search space with variations both among the receptors and the ligands. Thus, it is possible to model ligand affinities to receptors, and also - in some cases - to discern which parts of the protein that is most important for binding. I.e., with proteo-chemometric analysis it is sometimes possible to determine the molecular mechanisms in the interactions between ligands and receptors[1].

[11] Hibbert and James 1987.

2.5 Bioinformatics, Genomics, and Proteomics

To conclude the introduction of the application domain it is probably useful to discuss a few recent "buzz-words".

The *genome* is the total genetic information carried by a cell or organism. *Genomics* is the study of the structure and function of whole genomes or other very large collections of genes.

Even though we may have direct knowledge of all gene sequences for a given organism this does not imply that the functions of all such genes are understood. A new fundamental concept called *proteome* (PROTEin complement to a genOME) has emerged that should drastically help genomics to unravel biochemical and physiological mechanisms at the molecular level[12]. The proteome is the set of all proteins synthesised in a given organism, and *proteomics* is the identification of the complete set of proteins synthesised by a cell under a given set of physiological and environmental conditions, and the determination of the proteins' roles in cell activities[13]. Genomics may be divided into the subfields structural and functional genomics, and proteomics is in turn a subfield of functional genomics.

The next level in the hierarchy of concepts is "physiomics", which considers how the proteome within and among cells co-operates to produce the biochemistry and physiology of individual cells and organisms[14].

Life science data is a very broad concept, encompassing proteomic and genomic data, as well as - among other things - clinical, toxicological, and chemical data. All these kinds of data are needed for drug development in the pharmaceutical industry.

IBM recognises a series of "challenges" which need to be met for a successful use of life science data[15]: (i) Integration of increasing and diverse data sources. (ii) Integration across functional "silos" within the R&D organisation. (iii) Knowledge management, sharing and collaboration. (iv) Data management, security, access, and storage management. (v) Business-to-business integration for outsourced functions. The PAQS project will mainly consider points (i) and (iii) of these.

Bioinformatics

The consequence of the breadth and scale of research efforts in structural and functional genomics is a very large quantity of data. A new discipline called "bioinformatics" has evolved, which deals with moving the data into (relational) databases, and with developing efficient methods for searching and viewing these data[14]. While genomics and proteomics heavily relies on the use of databases the research is mainly conducted in "white biology" laboratories. On the other hand, bioinformatics is purely "in-silico biology".

One biology textbook (Lodish et al 2001) defines *bioinformatics* as "the rapidly developing area of computer science devoted to collecting, organising, and analysing DNA and protein sequences". Denning (2000), former chair of the Association of Computing Machinery (ACM), writes that bioinformatics is "an emerging area of intimate collaboration between computing and the biological sciences. Investigators are exploring a variety of models and architectures that can revolutionise computing, biology, and medicine". Denning also divides

[12] *What is Proteomics?* 2001-11-30

[13] Butt 2001.

[14] *What is Genomics?* 2001-11-30.

[15] *IBM Life Sciences Framework* 2001-12-13.

computer science into 12 subareas, 11 established[16] and one emerging - bioinformatics. Within the bioinformatics community, one opinion is that bioinformatics is "a new, growing area of science that uses computational approaches to answer biological questions" (Baxevanis 2001b). As in many scientific fields, the question is whether to focus on the methods or on the results. In bioinformatics there is still a need for both. In practice, bioinformatics applications are typically *used* by biologists since the domain knowledge is needed to interpret the results of database searches and application program computations.

An interesting question is whether the PAQS project should be considered to be bioinformatics research. PAQS deals with the analysis of interactions between proteins and ligands, and it relies both on advances in database research and multivariate statistics. Hence, PAQS clearly lies within the field of "informatics", and it also deal with biological entities on a molecular level. In my opinion it is clear that the PAQS project lies within the domain of bioinformatics.

[16] E.g.: Algorithms and data structures; Databases and information retrieval; Human computer interaction.

3 Databases - an Introduction

In this chapter an introduction to the area of databases and database-related concepts will be given, intended mainly for non-computer scientist readers. The chapter mainly consists of "standard" text-book material[17], except for the last subsection (3.2.6) which is specialised on topics specific for this Thesis.

3.1 Databases, Database Systems, and Database Management Systems

There are numerous explanations and definitions of what a database actually is. Definitions range from the quite general "a collection of related data" (Elmasri and Navathe 2000) to more restrictive ones. Elmasri and Navathe lists three implicit properties for a database: (i) A database represents some aspect of the real world, i.e. some aspect that is of interest for users; (ii) A database consists of a collection of data which is logically coherent and has some inherent meaning; (iii) A database is designed, built, and populated for a specific purpose.

Usually, when people use the term "database" they mean something that is stored electronically (on disk, on tape, on in a computer's main memory), and is managed by a specialised software, a database management system (DBMS, vide infra). This is also what "database" most often will mean in this Thesis. In some places, however, we may use the term to encompass also other data sources on the Web.

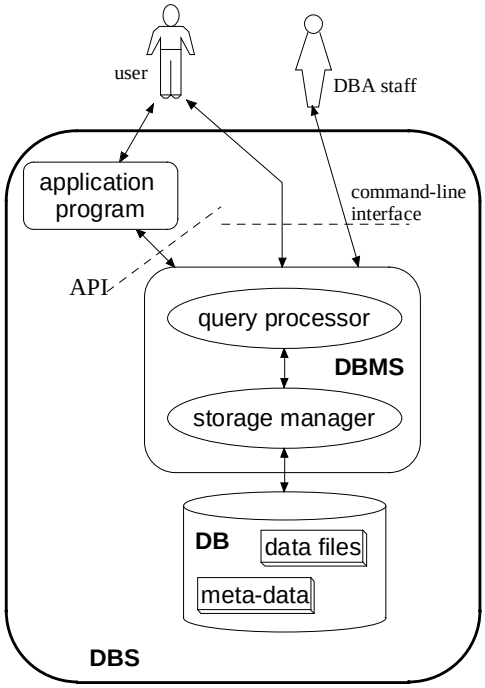


Figure 2. A database system (DBS) consisting of a stored database (DB), a database management system (DBMS), and an application program. Users interact with the DBS either by issuing queries directly through the command-line interface or by executing the application program. An application programming interface (API) serves as a link between the DBMS and the application program. The DBS is maintained by database administration (DBA) staff.

[17] Elmasri and Navathe 2000; Connolly and Begg 2002.

Figure 2 relates the concepts database (DB), database system (DBS), and database management system (DBMS). To exemplify: The protein information resource[18] (PIR) is a database system consisting of various application programs (e.g. web forms and programs to let users retrieve data), an Oracle DBMS, and the data which resides in disk files. Although we may informally refer to the PIR as "an Oracle database" it would be more correct to say that the database is managed by an Oracle DBMS, or that the PIR data resides in a database system of which an Oracle DBMS is one important component.

A DBMS (DataBase Management System) is a specialised software package for managing databases. There are a range of tasks a DBMS should fulfil. The most obvious task is that a user should be able to access (retrieve and manipulate) data stored in the database. A few other tasks are (i) to provide facilities for backups and for restoring data after a system failure, (ii) to impose constraints such that the database is always in a consistent state (e.g. to ensure that all employees have different employee numbers), (iii) to provide authorisation control, and (iv) to allow many users to simultaneously use the database. Obviously, the DBMS should also provide some mechanism for constructing the database in the first place, and for modifying its structure.

3.1.1 Metadata and Program-Data Independence

An important aspect of a database system is that a description of the structure of the database is stored in the database itself. Such information is called *metadata* ("data about data") and is used by the DBMS in its work to access the data files. In a relational database (vide infra) the names of tables and columns are examples of such metadata.

If we use an application program that directly reads and writes data from/to a data file on disk (no DBMS involved) the structure of the data file is explicitly embedded in the application program[19]. Each time we change the structure of the data file[20] we need to make the corresponding changes to the application program. On the other hand, if we use a DBMS any changes of the data file structures can be absorbed by mappings within the DBMS, and the application program will not have to be changed. Thus, the use of a modern DBMS provides *program-data independence*[21].

3.1.2 Access Methods

Once the data are stored in a database there must be some fast way to get hold of the data. Here, a few *access methods* will be described. No practical work on access methods has been made for this Thesis, but an introduction to the concepts is important in order to get an

[18] The Protein Information Resource is a division of the National Biomedical Research Foundation (Washington DC), see http://pir.georgetown.edu. The PIR-PSD (protein sequence database) is claimed to be the largest public domain protein sequence database in which entries are annotated and classified (Barker et al 201).

[19] For example, a Fortran program reading a direct access disk file with employee records would have specified the length of each record in an OPEN-statement, and the order, length and type of each field (employee number, name, employed_since, salary) in READ- and FORMAT-statements.

[20] E.g., if we add a field "department" to the employees, or simply change the length of employed_since from six to eight digits.

[21] Text books on databases will further describe logical and physical data independence as consequences of the ANSI-SPARC three-schema architecture. These are not our concern here.

understanding of future relevant subprojects for PAQS, and what various data sources can and cannot do.

For the discussion, suppose we have a file of persons, each with a name, an age, and a title. In this example the name is unique for each person, while age and title are not. In an *unordered file* data are stored in random order, for example in the order they were entered. If there is no accompanying access structure we need to scan through the file until we find the right person[22]. In many commercial relational DBMSs (vide infra) this is called a "table scan".

An *index* is some data structure which helps the DBMS find the correct record fast. There are different kinds of indexes, suitable for different purposes. Presently B-tree indexes and hash indexes are implemented in Amos II. For example, a search for the person with name 'Fredrik' will be fast if there is an index on the persons' names (either hash or B-tree), and a search for all young persons (age < 30, a "range query") will be fast if there is a B-tree index on the persons' ages[23].

For "modern" database problems it is often necessary to employ other index types than those mentioned above. E.g., geographical data is often indexed by R-trees, and text retrieval systems use "inverted indexes"[24]. In order to index chemical substances according to which functional groups they contain bitmap indexes can be used[25]. Sequences of characters (e.g. the genetic code or the amino acids of a protein) can also be indexed, for example by suffix arrays[26].

When we discuss information integration in chapter 4 we will see that one argument for the mediator/wrapper approach of Amos II (and against data warehousing) is that the data sources may have unique capabilities for their specific kind of data. These capabilities can only be utilised when the data resides in the original sources, not when it has been copied to a central repository.

3.2 Data Models and Database Schemas

A *data model* is a particular way of describing data, relations between the data and constraints on the data[27]. Another way to express this is that "a data model is a collection of concepts used to describe the structure of a database" [28]. Usually, the data model also includes some basic operations to retrieve and update data in the database[28].

A *database schema* on the other hand is the description of the database, made in the language of a chosen data model.

[22] Obviously, the file could be kept ordered on one field, e.g. the persons' names. Then, when we look for 'Fredrik' we don't need to scan through the whole file, but can use a method called "binary search". However, keeping the file sorted will require a lot of work if persons are inserted and deleted.

[23] These and other index data structures are described in most text books on databases, see, e.g., Elmasri and Navathe 2000.

[24] Garcia-Molina, Ullman and Widom 2000.

[25] A "fingerprint" is a special kind of bitmap index suitable for screening large chemical databases, see, e.g., James, Weininger and Delany 2000.

[26] Andersson 2000.

[27] Connolly and Begg 2002, p 817.

[28] Elmasri and Navathe 2000, p 24.

As mentioned in the previous section, a DBMS is a package of computer programs, used to manage a stored database. It is worth noting that a DBMS implements a specific data model, and that the stored database is a "population" (or extension) of a specific database schema (the intension). The prototype of PAQS presented in this Thesis has been constructed with Amos II, a DBMS implementing a variant of the functional data model. In chapter 8 we discuss various alternative solutions to details in the database schema, but we will not depart from the functional data model.

Although the work presented here has been made in the functional data model the following subsections will briefly mention other wide-spread data models and a few diagrammatic notations. This will hopefully provide a useful basis for readers not familiar with database terminology when comparing the approach of this Thesis with other work.

3.2.1 The Entity-Relationship Data Model

The Entity-Relationship (ER) model and the enhanced ER (EER) model are well-known examples of *conceptual data models*. These are used to construct high-level, "conceptual" database schemas which ordinary users relatively easily can understand.

In the ER model the information domain is described in terms of entities, attributes, and relationships. An entity is an object we are interested in, e.g. a person, and an attribute is some value we ascribe to that person (e.g. the name "Fredrik"). An entity set is a collection of entities with similar structure (e.g. all persons), while an entity type is the formal definition of what entities belonging to a specific entity set should look like. A relationship is some association between one entity and another entity, and as for entities we also have relationship sets and relationship types.

Typically, conceptual data models use a diagrammatic notation. For example, Figure 3 shows a simple ER diagram for a database schema with two entity types, four attributes, and one relationship type.

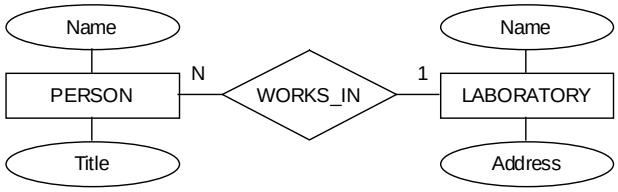


Figure 3. ER schema diagram showing the entity types PERSON and LABORATORY, each with two attributes, and the interconnecting relationship type WORKS_IN. Each "real person" is represented by an entity, i.e. an instance of the entity type PERSON, and so on. The cardinalities 1 and N signify that a person may only work in one laboratory, while a laboratory may have several people working in it.

3.2.2 The Relational Data Model

Since conceptual data models use high-level concepts, suitable for discussions between database designers and users, schemas made within these models are usually not well suited for implementation.

The *relational data model* and the legacy network and hierarchical data models are the most well-known *representational data models*, and may be implemented in a direct way, contrary to the conceptual data models.

In the relational data model the database is logically a collection of relations (tables). A relational schema defines a relation as a set of attributes (columns of the table), and the relation itself consists of a set of tuples (rows) with values. Each tuple should have a simple ("atomic") value for each attribute, i.e. there should be a single data value in each "slot" of the table. The relational model has a strong mathematical foundation and is easy to implement. Furthermore, the mapping from a schema in the ER model to a table description in the relational model is straightforward, see Figure 4. (A more detailed relational schema would also contain the data types of the attributes.)

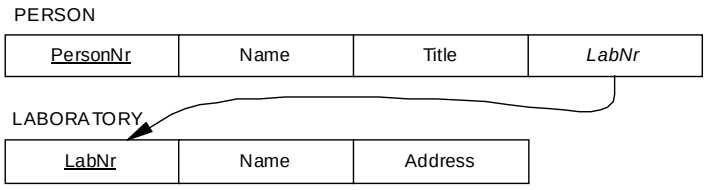


Figure 4. Relational schema diagram. Underlined attributes are primary keys (unique) and the arrow denotes the constraint that each value of PERSON.LabNr has to be an existing value of LABORATORY.LabNr.

The relational data model has been the dominant data model for the last 20 years and a few examples of commercial relational DBMSs are Microsoft Access[29], Oracle 7[29], and IBM's DB2 v5[29]. Later versions of Oracle and DB2 have been extended with object-oriented features, and may now be called object-relational DBMSs (vide infra).

SQL

While it seems straightforward to implement the PERSON relational schema of Figure 4 as a `struct` in the programming language C, for example, this is not something users or administrators of database systems need to do. Instead, they use SQL (the Structured Query Language):

```
create table PERSON (
    PersonNr int primary key,
    Name varchar(50),
    Title varchar(20),
    LabNr int references LABORATORY(LabNr)
);
```

Most users come in contact with SQL as a query language, to retrieve data from the database, and possibly they also insert, delete and update data. These operations all use SQL as a data manipulation language (DML). Conversely, when a new table is created or a new column is added to an existing table, SQL is used as a data definition language (DDL). The version of SQL used together with the DBMS Amos II is called AMOSQL, and this language too contains primitives for data manipulation as well as data definition.

SQL is a declarative query language, which means that we only need to specify *what* we want to retrieve, not *how* it should be retrieved. For example, to get hold of the titles of all persons named 'Fredrik' the following SQL query would be issued:

```
select Title from PERSON where Name='Fredrik';
```

Should we write the same in C or Java, we would need to first open the file PERSON, then scan the file for all records with the correct name, and finally print the title fields of the matching records. This is all made transparently by SQL, and we see one advantage of using a

[29] Registered trademarks of Microsoft Corporation (http://www.microsoft.com/office/access), Oracle Corporation (http://www.oracle.com), and International Business Machines Corporation (http://www-4.ibm.com/software/data/db2), respectively.

DBMS to handle stored data. Furthermore, since we do not specify *how* the result should be obtained the DBMS is free to perform a range of optimisations in order to retrieve the answer quickly.

Amos II is a main-memory database, and we can see one advantage of the database approach if we compare a running Amos II system with a run-time system implemented in Java. (The comparison of the previous paragraph referred to data stored on disk.) Both systems would store real-world concepts as objects (e.g. of type `Person`), but there is a significant difference in how we could get hold of the objects we want. In a pure Java application, users *navigate* from object to object. This is possible in an Amos II system too, but in Amos II we also have access to a powerful declarative query language and can use this to find relevant objects[30].

3.2.3 Object Data Models

The original incentive for developing object-oriented database management systems (OODBMSs) was to provide a means to transparently store objects of application programs written in object-oriented programming languages such as C++ and Smalltalk[31], i.e. to facilitate object persistence. By using an object-oriented approach both in application program and database the so-called "impedance mismatch problem" could be avoided.

A more important argument, particularly from the perspective of the PAQS project, is that OODBMSs supposedly are more suited than RDBMSs for handling "modern applications", such as computer-aided design (CAD), computer-aided software engineering (CASE), geographical information systems (GIS), and multimedia systems[32]. Some factors in favour of OODBMSs are that they have a user-extensible type system, and that they more easily than RDBMSs can handle complex object, long transactions, object versioning, and schema evolution. Three of the major commercial OODBMSs are Objectivity/DB[33], ObjectStore[33], and FastObject t7[33].

In an object data model, data are seen as objects with properties, or state, and behaviour. We may further speak of object types (vide infra) as a kind of user-defined data types. In an object-oriented (OO) environment the "real world" is modelled by a collection of objects that communicate with each other by exchanging messages. An important difference relative the relational model is that each object has a system-unique object identifier. Another difference, important from a modelling perspective, is that the implemented database schema often is easier to understand.

Some important features of object-orientation (e.g. inheritance, overloading, and polymorphism) will be discussed in section 3.2.6, in connection with the Amos II data model.

The Object Data Management Group

One major drawback with OODBMSs relative RDBMSs is the lack of a standard. Different OODBMSs implement slightly different object-oriented data models (or object data models for short). The Object Data Management Group (ODMG) is a consortium of leading

[30] In order to simulate database functionality a Java application could put objects in a hash table "registry". The objects would then be easily retrieved by the hash key (e.g. `Name`).

[31] While Java is arguably the most important object-oriented language, it did not exist at the time (end of 1980's).

[32] Elmasri and Navathe 2000; Connolly and Begg 2002.

[33] Registered trademarks of eXcelon Corporation (http://www.objectdesign.com/index2.html), Objectivity, Inc. (http://www.objectivity.com), and Poet Software GmbH (http://www.fastobjects.com), respectively.

OODBMS vendors, and the major body of standardisation for OODBMSs. Their Object Standard[34] consists of an object model, an object specification language, an object query language (OQL), and bindings to C++, Smalltalk, and Java.

In the ODMG object data model the state of an object is defined by the values of its properties, and the properties are either attributes or relationships. The behaviour of an object is defined by the set of operations that can be executed on or by the object. Objects with a similarly defined set of properties and operations can be said to belong to the same class. An alternative view, held by the ODMG object model, is that a class is a specification of the abstract state and behaviour of an object type. Thus, these two uses of the concept class correspond to entity set and entity type in the ER model.

The ODMG object model puts two aspects of the definition of a *type*: An external specification, and one or more implementations. An *interface* specifies the abstract behaviour of an object type, while a class specifies both abstract behaviour and state. Thus, in the ODMG object model we only need to know the interface of a type to be able to discuss how objects of that type may interact with other objects, but the class is necessary for a definition of the database schema. The functional model used in Amos II (section 3.2.6) is fairly close to the ODMG object model, but differs from it in some important ways - for example, there are (formally) no attributes in Amos II.

OQL - the Object Query Language

OQL (the Object Query Language)[34] is the query language associated with ODMG's data model. It is in many aspects similar to the standard SQL92 for relational DBMSs, but includes OO features such as object identity, complex objects, and polymorphism[35].

Two other object query languages are OSQL and AMOSQL, of the Iris[36] and Amos II systems, respectively.

UML - the Unified Modelling Language

The Unified Modelling Language (UML)[37] has during the last five years become a de facto standard for object-oriented analysis and design. Although the diagrams of this Thesis will be drawn in an ER-diagram style (similar to Figure 3) it is worth noting that UML will probably soon become dominant also for database modelling[38]. Figure 5 shows a UML class diagram corresponding to the ER schema in Figure 3[39]. (The operation `changeLab` was added in order to show how operations of OO classes are represented.)

[34] Cattell, Barry, Berler, Eastman, Jordan, Russel, Schadow, Stanienda and Velez 2000.

[35] Thus, the objectives of OQL and the new SQL:1999 standard (Eisenberg and Melton 1999a) are very similar. Eventually, they may be merged to a single standard, or OQL may be "buried" by SQL:1999 (Stonebraker and Brown 1999).

[36] Wilkinson, Lyngbæk and Hasan 1990.

[37] A short and very good text on UML is Fowler 2000.

[38] Two recent books on UML for database modelling: *Uml for Database Design*, E.J. Naiburg, R.A. Maksimchuk, Addison-Wesley (2001); *Oracle8 Database Design Using UML Object Modeling*, P. Dorsey, J.R. Hudicka, McGraw-Hill (1998).

[39] Since the UML diagram is meant to have a conceptual perspective (Fowler 2000) it doesn't specify data types. For the same reason encapsulation is not considered.

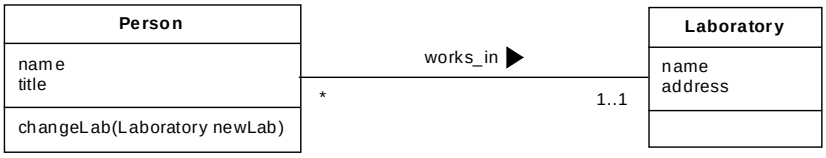


Figure 5. UML class diagram with two classes, four attributes, one operation, and one association.

In the terminology of UML, relationship types (between entity types) are called associations (between classes), relationships between entities are links between objects, and cardinalities are multiplicities.

Two Additional Advantages of Object-Oriented Databases

It is often easier to understand an implemented object-oriented database schema than a relational database schema. Due to a process called "normalisation" data pertaining to the same concept (e.g. "person") may be spread over several relations (tables) in the relational schema. Due to the possibility to include lists, sets, bags (multisets) and other data structures as attributes to an object, the corresponding OO schema will usually have relatively coherent object types.

Finally, we may note that during database design the object data model has a wider scope than the relational or the ER data models. In the "traditional" approach to database design a conceptual database schema is first constructed as an ER diagram, followed by mapping to a "logical" database schema in the relational model. On the other hand, with an OO approach we may use the OO data model both for conceptual and logical schemas. The same is true for the functional data model (3.2.6), which was used in this Thesis.

3.2.4 Object-Relational DBMSs

Most of the major relational DBMSs have lately been converted to object-relational (OR) DBMSs, which means that desired object-oriented features have been incorporated into the relational products. For example, Oracle8i[40], IBM's DB2 UDB[40] (universal database server), and Informix Dynamic Server with Universal Data Option[40] (IDS-UDO) may all be termed object-relational[41].

Stonebraker and Brown (1999) list four main features of an ORDBMS: support for (i) base type extensions, (ii) complex objects, (iii) inheritance, and (iv) a production rule system. The first three features must be available in an SQL context. It is worth noting that Amos II, the DBMS used in this work, sometimes is referred to as being object-relational. All four features above exist in Amos II, and features (i)-(iii) are probably used by most Amos II applications. The context, however, is not standard SQL but AMOSQL.

An important aspect of commercial ORDBMSs is their abilities to use "plug-ins" to extend the functionality for a particular information domain. For example, there are spatial extensions available for all three ORDBMS products mentioned above, e.g. suitable for

[40] Registered trademarks of Oracle Corporation (http://www.oracle.com) or International Business Machines Corporation (http://www-4.ibm.com/software/data/db2 and http://www-3.ibm.com/software/data/informix). Informix was recently acquired by IBM, but IDS-UDO remains a distinct product.

[41] Stonebraker and Brown (1999) discuss various strategies a relational DBMS vendor may use to produce an object-relational DBMS. IBM and Informix have constructed "object-relational tops" on relational storage managers, while Oracle has used a strategy of incremental evolution of the relational DBMS.

efficient storage, access and analysis of GIS application data. These kind of extensions are called DataBlades (Informix), Cartridges (Oracle), and Extenders (DB2 UDB). In Amos II this functionality is achieved by means of foreign functions. The new SQL:1999 standard also includes external routines[42].

Often these kind of "plug-in" products are developed by independent third parties. For example, DayCart[43] is a "bundled set of tools which extends the Oracle server with new chemical capabilities" [44]. To my knowledge, no such extension presently exists for the bioinformatics area. Since bioinformatics has been a "hot topic" for the last ten years this seems surprising, and I assume it is only a matter of time until the first one appears on the market.

3.2.5 Functional Data Models

In functional data models entities and functions are used to represent real world objects and properties of those objects, respectively. If we compare with the object model we see that functions take the roles of both attributes and relationships.

For database schemas constructed in a functional data model a diagrammatic notation similar to that of ER-diagrams can be used, see Figure 6.

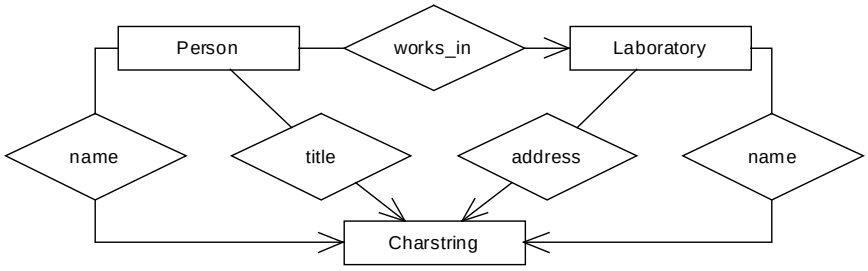


Figure 6. Functional database schema diagram with three entity types and five functions.

A function maps a given entity onto a set or a bag of target entities[45], and hence there is always an explicit direction drawn for a function. As described in more detail in the next section, we may add constraints on this mapping, e.g. we may require that the function maps an entity onto a single (unique) entity, or that the target is an ordered sequence of entities.

Many discussions of the functional data model for databases take Shipman's (1981) work on the functional model and the DAPLEX language as a starting point. This DAPLEX model[46], and other "semantic data models" were meant to represent "the real world" more closely than previous data models[47]. As the benefits of the object-oriented approach became more and

[42] Eisenberg and Melton 1999b.

[43] Registered trademark of Daylight Chemical Information Systems, Inc. (http://www.daylight.com/products/daycart.html).

[44] Kappler and Delany 2001.

[45] In the original DAPLEX functional data model (Shipman 1981) a function maps an entity onto a *set* of entities (no duplicates allowed). A *bag* (or multiset) is an unordered collection of objects where duplicates are allowed.

[46] Shipman (1981) uses the term "DAPLEX" to denote a data definition and data manipulation language. However, "the DAPLEX model" is often used to denote both the language and the underlying functional model as it is described by Shipman. This is also how the term is used in this Thesis.

[47] Connolly and Begg 2002, p 807.

more apparent, the DAPLEX model was extended by different research groups to incorporate various OO features. The database schema presented in this Thesis has been implemented in the Amos II system[48]. Several other research DBMSs implement some form of functional (or functional/object) models, e.g. P/FDM (section 6.1.1), Multibase[49], Pegasus[50], and Iris[51].

3.2.6 The Functional Data Model of Amos II

Amos II implements an object-oriented extension of the DAPLEX functional model. The Amos II data model is based on the IRIS data model[52], and has three main concepts: objects, types, and functions. Similarly to Shipman's DAPLEX model an Amos II function maps an object onto one or several objects, but the concept of type is new. From now on we will deal with the Amos II data model (Risch, Josifovski and Katchaounov 2000).

Diagrammatic Notation

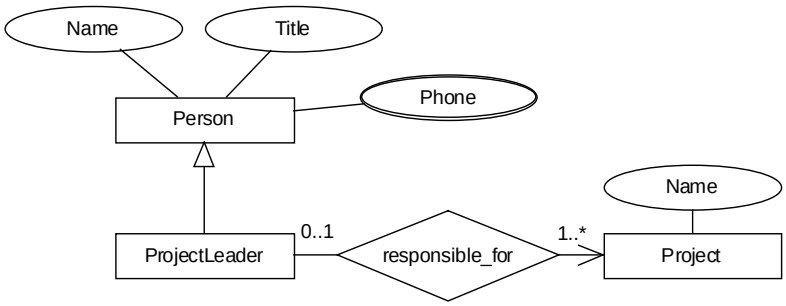


Figure 7. Database schema in the functional model, drawn in the diagrammatic notation used in this Thesis.

Figure 7 exemplifies how schema diagrams will be drawn in the rest of this Thesis. Symbols for types (rectangles) and functions (diamonds, ovals) were taken over from the ER diagram notation. In most diagrams a function which maps a type onto a literal type (vide infra) is represented by an oval while a diamond represents a function that maps a type onto a user-defined type. Ovals with double borders correspond to "multi-valued attributes" of the ER model (e.g., in Figure 7 a person may have several phone numbers, but only one name).

Symbols for inheritance (arrows with unfilled heads) and cardinalities (0..1) are the same as in the UML. Cardinalities follow a min..max notation, as in the UML and some versions of ER modelling[53]. Thus, the cardinalities of Figure 7 mean (a) that a project leader is responsible for at least one project, and (b) that a project has a single leader, or none.

The "open" arrowhead from a diamond to a rectangle denotes the direction of the function.

---

[48] Risch, Josifovski and Katchaounov 2001; Risch and Josifovski 2001.

[49] Landers and Rosenberg 1986

[50] Ahmed, De Smedt, Du, Kent, Ketabchi, Litwin, Rafii and Shan 1991.

[51] Wilkinson, Lyngbæk and Hasan 1990.

[52] Lyngbæk and Kent 1986.

[53] We would say that a person has 4..4 biological grandparents, but if we were only interested in relatives which are alive we would make it 0..4 grandparents. A * denotes any natural number. A few combinations of min..max are so common that we don't need to write them out in full: Only 1 means 1..1 (mandatory, an object of this type *must* take part in a function of this type), and only * means 0..*. Two other very common situations, without special symbols, are 0..1 and 1..*.

Objects and Types

There are two kinds of objects in Amos II: literals (integers, reals, strings, et cetera) and surrogates. Only objects of surrogate types have object identifiers (OIDs)[54]. Typically, "real world" objects we are interested in are represented as Amos II surrogate objects. For example, the object :donald[55] would be of the surrogate type Person, while the literal "Donald" represents his name and the literal 30 his age.

Further, there are four kinds of surrogate types: stored, proxy, derived, and integration union types. Objects of a stored type are created by the user, and are stored locally. The latter three surrogate types are used for information integration, and will be used in a demonstration example (see Appendix G).

Types and Inheritance

Each object belongs to a type, and as in the OO approach a type may be part of a generalisation/specialisation hierarchy. E.g., we may have the type Person and another type ProjectLeader. Obviously, a project leader *is a* person, and all functions we have defined for the type Person are relevant and necessary also for the type ProjectLeader. On the other hand, the type ProjectLeader may need a new function "responsible_for", mapping onto a Project object. Thus, ProjectLeader is a *subtype* (or specialisation) of Person[56]. The situation is showed diagrammatically in Figure 7.

The fact that ProjectLeader is a subtype of Person means that all functions defined for Person are *inherited* by ProjectLeader. We save some work in not needing to redefine them, but the three big benefits of subtyping is that (1) we can express more of the "real-world" semantics when we model, (2) the database schema gets easier to understand, and (3) we may use a ProjectLeader in every place we can use a Person[57]. The converse to (3) is of course not true. If we try to invoke the function responsible_for on an object which is of type Person but not ProjectLeader the system will generate an error.

A constraint put on the type system in Amos II is that an object must always have a most specific type. I.e., if ProjectLeader and Professor are two subtypes of Person, we cannot make :donald a member of both those types since they are equally specific[58]. The solution here is that we construct a new type ProjectLeadingProfessor as a subtype of both ProjectLeader and Professor. This situation, when a type inherits directly from two different supertypes, is called *multiple inheritance*[59]. Obviously, this solution gets messy when there are more than two overlapping subtypes, and it is furthermore easy to forget some subtypes when the schema evolves. In these cases, other design solutions should be used, e.g. delegation instead of inheritance[60].

[54] Literals and surrogates correspond to literals and objects of the ODMG object model (Cattell et al 2000).

[55] The colon in the beginning of :donald denotes that we refer to a specific object. The variable :donald is a reference to the object.

[56] From a technical point of view it is possible to create two types Person and Project, and then give them a common subtype ProjectLeader. However, this is a major modelling error since we cannot say that a project leader *is a* project!

[57] An object which belongs to type *t* does also belong to all supertypes of *t*.

[58] When :donald is a member of both ProjectLeader and Person it is clear that ProjectLeader is the most specific type since it is a subtype of Person.

[59] Multiple inheritance is allowed in some OO programming languages (C++) but not in others (Java).

[60] Grand 1998.

An object may dynamically change type, or take on a new type. For example, imagine that Donald, represented by an object :donald of type Person, is appointed project leader of a new project. Then we may add the type ProjectLeader to :donald, and when the project is finished, we may remove this type[61]. In Amos II this dynamic addition of types to an object is only possible when the new type is a subtype of the object's most specific type.

Finally, a set of subtypes may be disjoint or overlapping. E.g., if the type Person had the two subtypes Man and Woman these would be disjoint[62]. Conversely, two subtypes Professor and Woman would be overlapping[63].

Functions

As in the DAPLEX model, the Amos II functions describe both properties and behaviour of an object. There are five kind of functions in Amos II:

- A *stored function* represents an attribute or a relationship which is stored in the database. Stored functions resemble tables in the relational model.
- *Derived functions* are defined as AMOSQL queries over other functions. The corresponding construct in the relational model is a view.
- *Foreign functions* are implemented externally of the Amos II system, as Java, Lisp, or C programs. They can be used to make complex computations, and thus correspond to "data blades", "cartridges", or "extenders" of ORDBMSs. Another use of foreign functions is to implement graphical user interfaces[64].
- A *proxy function* represents a function of some other database, and may be used for mediation.
- A *database procedure* has the signature of a normal function, but is implemented in a procedural sublanguage of AMOSQL.

Stored and derived functions are used all through the PAQS prototype. In several places, foreign functions and database procedures are used.

Overloading of Functions

The functions name(Person) -> charstring and title(Person) -> charstring are obviously different functions, and any system can distinguish between them. Amos II, like object-oriented systems, support *overloading*, which means that the system understands name(Person) -> charstring and name(Laboratory) -> charstring as different

[61] In Java an object always has the same type.

[62] The specialisation of Person into Man and Woman is furthermore *total*, or *exhaustive*, and the extents of Man and Woman together constitute a *partition* of the extent of Person (Boman, Bubenko, Johannesson, Wangler 1997). In UML a total specialisation can be represented by abstract classes. We will use this in section 8.8.3 and Appendix E.

[63] In the diagrams I will use the convention that subtypes which are attached to the same arrowhead symbol are disjoint, other overlapping (see, e.g., Figure 34, where a BindingExperiment is either a TimeBindingExperiment or a ConcBindingExperiment). Alternatively, constraint labels of the kind {disjoint} can be used.

As described in the previous paragraph, implementation of a schema with two overlapping subtypes in Amos II requires the introduction of a common subtype.

[64] E.g. "goovi" of the Amos II system (Cassel and Risch 2001).

functions. The system "resolves" the name ambiguity between the two functions by means of the types of their arguments, `Person` and `Laboratory`[65].

However, to overload the functions `name(Person)` and `name(Laboratory)` is mainly a matter of convenience, we don't need to invent (and remember) a new function name. To get an example of where overloading matters more, we use a function `personalData(Person)`. The personal data could for example be needed for a web server which displays information about all persons working at a laboratory. The personal data for a `Person` would be name and title, but for a `ProjectLeader` it would be name, title, *and* the names of all projects he or she is responsible for. The difference from the previous example is that we will require the system to take a list of persons working at the laboratory, where *some* persons are project leaders but most are not, and print the personal data. By means of overloading this is possible: For most persons in the list only name and title is printed, but if the `Person` is also a `ProjectLeader` the implementation of `personalData(ProjectLeader)` will take precedence, and the names of the projects will be printed together with name and title of the person. This is an example of *polymorphism*, a central feature in the object-oriented approach.

Cardinality Constraints

The only cardinality constraints built into the Amos II functional model are uniqueness constraints:

- If we declare a function `name( Person key ) -> charstring nonkey` we know that the function `name` maps a `Person` object onto a single `charstring`. I.e., a person has a single name. (However, we have not constrained names to be unique, i.e., there may be several persons with the same name.)
- If we declare a function `phones( Person nonkey ) -> charstring nonkey` a `Person` object may have several phone numbers. Equivalently, we may declare the function as `phones( Person ) -> bag of charstring`. I.e., the function `phones` may return several strings, and several persons may share a phone numbers.
- We may also make the target of the function unique, as in `name( ChemicalSubstance nonkey ) -> charstring key`. This is reasonable if we wish to store many alternative names for each chemical substance, while keeping these names unique. E.g., the names chloromethane and methyl chloride refer to the same specific chemical substance.
- Finally, if the mapping is 1:1, we make both sides of the function keys: `employeeID( Employee key ) -> charstring key`.

The default is to make the first argument of a function a key, and all other arguments plus the target non-keys. Thus, the first example in the list could also be written `name( Person ) -> charstring`.

There is no direct way of setting other cardinality constraints, e.g. to constrain a person to have 0..2 biological parents. However, we may accomplish this by always using a database procedure when adding parents, and to let this procedure check that the person doesn't get too many parents[66]. Alternatively, we could use the active rule system of Amos II. In this specific

[65] Although the type of the result is formally a part of the function signature, the Amos II system cannot resolve overloading on result types, and neither can OO languages such as Java or C++ system. Thus, the system cannot distinguish between the functions `boss(Laboratory) -> Person` and `boss(Laboratory) -> charstring`.

[66] Unlike in Java and other OO programming languages, there is no special constructor functionality in AMOSQL.

example, the semantics suggest a third solution: We simply let the `Person` type have two different functions, `mother` and `father`, each giving single-valued results.

Ordered Sequences of Objects

The Amos II data type `vector` makes it possible to store objects (surrogates or literals) in an ordered sequence. This seems to be particularly useful for scientific and technical databases[67]. For example, in the PAQS prototype such sequences are used for data series (see section 8.2). However, there are presently quite few operations available for vectors (creation and access by index number). In order to get more functionality out of the useful data structure I have implemented a few vector manipulations as Java foreign functions, see Appendix C.

[67] Maier and Vance (1993) strongly advocates for the inclusion of ordered structures in scientific databases. They note that although many OODBMSs contain some list or vector data type these types cannot be used effectively in query processing. The same applies in Amos II.

4 Information Integration

One of the more important directions in recent research on database systems is information integration, i.e. to find methods by which data stored in two or more sources can be combined to one large, possibly virtual, database, and the combined data can be queried transparently by users and applications. The data sources may be a combination of databases, web sites, text files, et cetera.

4.1 Three Approaches to Information Integration

According to Garcia-Molina, Ullman and Widom (2000) the three main approaches to information integration are federation, warehousing, and mediation:

- *Federated databases* are independent, but one database knows how to call upon each of the others to get the information needed. Thus, for *n* federated databases up to *n(n-1)* translations are needed. As long as the databases only need limited functionality of each other a federated system may well be the easiest to build.
- A *data warehouse* may be used as a central repository for all integrated data in the organisation[68]. The data warehouse contains copies of data in the data sources, which are typically the organisation's production "on-line" databases. In contrary to most production databases, the warehouse contains historical data[69]. In order to provide fast answers to complex queries data may be presummarised at different levels (sales per day/month/year).

The warehousing approach has met with great success in the business world, and data warehouses and data marts[70] have been implemented as decision-support systems in many organisations.

- A *mediator* supports a virtual database, which the user may query transparently. The mediator stores little or no data locally. The concept of mediators was introduced by Wiederhold (1990, 1992), and the mediator/wrapper approach taken by Amos II and the PAQS project will be described further in section 4.4. It is possible to build a system of several interoperating Amos II servers, each wrapping one or several data sources. We then get a federated database system where each local database follows a mediator/ wrapper approach.

[68] A common definition is that a data warehouse is "a subject-oriented, integrated, time-variant, and non-volatile collection of data in support of management's decision-making process" (Connolly and Begg 2002, p 1047).

A few texts on Data Warehouses are Inmon, W.H. (1993) *Building the Data Warehouse*. New York, NY: John Wiley & Sons; Kimball, R. (1996) *The Data Warehouse Toolkit : Practical Techniques for Building Dimensional Data Warehouses*. New York, NY: John Wiley & Sons; Kimball, R. & Merz, R. (2000) *The Data Webhouse Toolkit: Building the Web-Enabled Data Warehouse*. New York, NY: John Wiley & Sons.

[69] For example, the inventory database of a shop should keep track of how many items of each product the shop has on stock right now, and it is important that this information is up-to-date. The data warehouse would keep a history of perhaps five years back in time in order to facilitate queries giving information on trends et cetera. The warehouse would also be a good data source for data mining applications searching for patterns in this sales history.

[70] Data marts are smaller than data warehouses, intended for a department of the organisation or for a special user group. The central idea of replicating data from the original data sources is the same as for data warehouses.

From the descriptions above we see that data warehousing differs from mediation and federation in one very significant way. With warehousing data from the sources are replicated in the warehouse, while in the other two approaches data always reside in the data sources. Typically, users cannot update data in the warehouse, i.e., the warehouse is read-only.

Obviously there are advantages and disadvantages either way. E.g., a data warehouse should respond faster to a query since all data needed resides in one place (less communication costs) and may even be preaggregated in some useful way. On the other hand, with federation and mediation data is always up-to-date (as long as communication lines are up), and there is no need for a huge central storage facility. Furthermore, data sources may keep a high degree of autonomy (vide infra), and they may have unique capabilities which cannot be exploited when the data has been taken over to the warehouse. As discussed by Hellerstein, Stonebraker and Caccia (1999) the warehousing approach breaks physical data independence as well as some aspects of logical data independence.

To conclude, we may say that warehousing is an approach used for data originating from within the organisation, possibly with some external "background" data added. If we mainly rely on data sources which are not under our control, warehousing is not suitable, and with full data source autonomy, e.g. all external data taken from web servers, mediation becomes the solution of preference.

4.2 Three Dimensions of Distributed Database Systems' Architectures

Architectures of distributed database systems (DDBSs) are often classified along three dimensions: distribution, heterogeneity, and autonomy (Özsu and Valduriez 1999):

- *Distribution* refers to where data is located.
- *Heterogeneity* refers to how different the local database systems are. They may be heterogeneous in a number of ways, some of the more important being hardware, data models, DBMSs (i.e. software), query languages, transaction protocols, and database schemas. Particularly, any attempt to integrate data from web servers and databases will have to deal with the quite varying data models, schemas, and querying capabilities of the data sources.
- Finally, *autonomy* refers to the distribution of control. For example, the local database systems may or may not be free (i) to make changes to the local database schema, (ii) to decide which parts of the local schema that other database systems may access, and (iii) to temporarily leave the distributed database system.

E.g., suppose that a distributed database system is developed top-down. Some central authority in the organisation decides that a DDBS is needed, what the database schema looks like, and which kind of hardware and software that will be used. Such a system will have the data distributed among the participating databases, but there will be no heterogeneity (all local database systems will look the same), and no autonomy (decisions are taken centrally, a local database administrator cannot decide to change the database schema, or to leave the DDBS). In this kind of system information integration should be a minor problem, it is provided for already in the design of the DDBS.

On the other hand, all three approaches described in section 4.1 are meant for systems designed in a bottom-up way: The data sources to be integrated already exist when the need for information integration arises, and they are most likely heterogeneous in several ways.

4.2.1 Heterogeneous Distributed Database Systems

Bouguettaya, Benatallah and Elmagarmid (1999) give an overview of heterogeneous distributed database systems, i.e. information systems that provide interoperation and some degree of integration among multiple databases[71]. The overall problem to solve is that of data integration: to provide a uniform and transparent access to the data managed by multiple databases. According to Bouguettaya et al this may basically be accomplished in three ways:

- After a *global schema integration* users get a uniform and consistent view of the data. Obviously, it may be difficult and time-consuming to completely integrate all local schemas. Furthermore, the method is not suitable if there are frequent changes to the local schemas.
- In a *federated database system* (FDBS) the local schema of each data source is translated into an export schema constructed in a common data model. A federated schema is created as a view against relevant export schemas, either by the user (decentralised, or loosely coupled FDBS) or by federation administrators (centralised, or tightly coupled FDBS).
- With a *multidatabase language* there is no need for a predefined global or partial schema. By means of this language users may query multiple databases at the same time. Some problems with this approach are that users need to know where data are located, and also understand the local schemas and "manually" resolve semantic conflicts.

Figure 8 shows a reference five-level architecture which is often used for discussing these types of systems. In the three approaches above there is one, several, and none federated/global schemas, respectively.

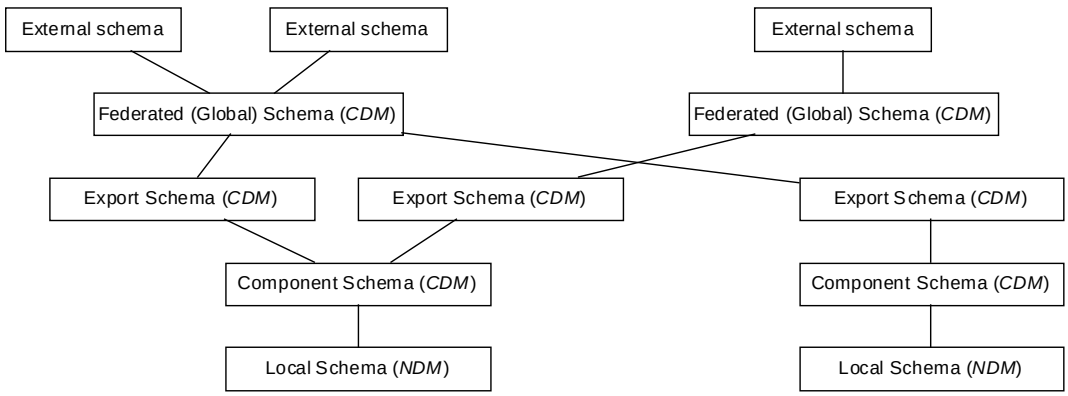


Figure 8. A five-level schema reference architecture for multidatabase systems (after Pitoura et al 1995). Local schema are constructed in the native data models (NDM) of the pre-existing database systems. When the multidatabase system is built the local schema are translated to component schema in the chosen common data model (CDM). A subset of the component schema is exported, and from the export schemas a federated or global schema is created by integration. Users and applications access the system through external schemas, which may be constructed in any data model.

A single Amos II mediator system which wraps several data sources works with a single schema. However, the wrappers usually only translate parts of the local schemas for use in the

[71] Other terms for heterogeneous distributed database systems are multidatabase systems and federated databases, see Bouguettaya et al (1999) for discussion and references. Note that the definition excludes the data warehousing approach as it assumes that data should remain localised in the data sources.

mediator, and the system would be a variant of a tightly coupled FDBS. Several Amos II servers may then be connected to form a federation, a loosely coupled FDBS.

4.3 Problems in Information Integration

In this section a short description is made of some problems that arise when data is integrated from different heterogeneous and autonomous data sources. The three architectural dimensions for distributed database systems (section 4.2) give rise to different kinds of problems.

4.3.1 Distribution of Data

Obviously, if we wish to integrate data from various *autonomous* sources there is little we can do about the distribution of data. In most cases we have no write access to the sources, and it is not realistic to load all source data to a local "warehouse". What we *can* do to increase performance is to cache frequently needed data locally in the mediator, and to store some precomputed summations and indexes[72] in the mediator.

4.3.2 Autonomy

Since the data sources are autonomous, we will need to comply with whatever data models, database schemas, and querying capabilities they provide. This can be solved by the wrapper approach described below, but a remaining problem is that external data sources may change schema without notifying us. Usually, such schema changes will be rather small, and the functional object model of Amos II is well suited for schema evolution in response to such changes.

A major problem is when a data source moves over to a new data format, e.g. from ASN.1 to XML (vide infra). Such a change requires that we construct a new wrapper for the source, and if we are quite unlucky (if the source is the only one to provide some important data) all queries towards the PAQS system will be impossible for several weeks[73].

Another autonomy-related problem is that it is quite possible, and probable, that some data sources where access is presently free of charge start to charge users. Different business models may give rise to different economical and technical problems.

4.3.3 Heterogeneity

The heterogeneity-related problem which probably has been studied most extensively is that of schema integration, where we need to consider both "the system level" (different data

[72] E.g., if we use four different web databases for protein amino acid sequences it might be useful to store an index over which proteins that are in which external databases. In this way, we would not need to query all databases. However, such an index needs to be kept up-to-date.

[73] A simple source may be wrapped in a week or two, while more complex sources may require up to a few months of work to be completely modelled (Haas, Schwarz, Kodali, Kotlar, Rice and Swope 2001). If we have wrapped the data source once, it will be easier the next time, and it will be particularly easy if we already know how to technically wrap the new format (e.g. a relational database or XML). However, when an organisation decides to make a change of data format, it is quite possible that they also change the schema drastically.

models have different constructs) and "the data level" (similar information represented differently) (Elmagarmid, Du and Ahmed 1999).

In general, the local schema of the data sources are translated to an integrated (partial or global) schema in a common data model. It is then clearly desirable that this common data model (e.g., the data model of the mediator) has greater modelling power than the data models of the sources, or else some semantics will likely be lost in the integrated schema. As mentioned in section 3.2.5, the functional model has rich semantic power[74], and several mediator DBMSs implement a object-oriented or functional object data model[75].

Some important possible mismatches between the schemas of the various data sources are[76]:

- *Identity conflicts*: The same "real-world" concept, e.g. a specific protein, is represented by different objects in different databases. This is really one of the things *we wish to integrate* in the PAQS project. As long as we can get the mediator to understand that the two objects refer to the same protein there is no problem, but what if the two databases have no unique protein identifier in common?
- A *schema conflict* arises when one concept is represented differently in the data source schemas. The difference may be in naming or in structure:
  - The same concept may be represented by different names (synonyms), or worse, different concepts may be represented by the same name (homonyms).
  - One concept may be represented by different data model constructs, e.g. the classification of some persons as professors may be implemented by subtyping or by a discriminator attribute (a "flag").
  - Data types may differ, e.g. phone numbers may be stored as character strings or integers.
  - Some concepts may be missing in one of the databases. E.g., one data source on affinity data may have no references to the original literature recorded, while all others do. This type of problem is often solved by using semi-structured data (e.g. XML) to represent integrated data which may not conform entirely (Garcia-Molina et al 2000).
- *Semantic conflicts*: The same concept is interpreted is (slightly) different ways in the various data sources. A range of possible conflict types exist, for example (Wiederhold 1982):
  - Scope: Are assistant professors really "professors"?
  - Abstraction: Does "income" refer to personal income or family income?
  - Time: Do we sum "total sales" over weeks or months?
- Finally, there are *data conflicts* where different databases store different values for the same concept (i.e., there is no semantic conflict). For example, one database may store the atomic weight of hydrogen as 1.008 and another as 1.0079. Data conflicts are relatively

[74] In a comparison of the functional, relational, network and hierarchical data models it was found that the functional model qualitatively subsumes the other three models (Demurjian and Hsiao 1988).

[75] Pitoura, Bukhres and Elmagarmid (1995) gives a compilation of object-oriented techniques for design and implementation of multidatabase systems.

[76] Pitoura et al (1995); Bouguettaya et al (1999); Garcia-Molina et al (2000); Wiederhold (1992).

easy to detect, and may be due to different levels of precision or to the fact that one database simply stores the wrong value[77].

In addition to the conflicts described above two schemas may, when taken together, have certain constraints in common. For example, a type in schema A might be constrained to be the subtype of a type in schema B[78]. Another example is when an attribute of schema A is derivable from one or several attributes of schema B[79]. Such correspondences are commonly called *interschema properties*[80].

4.4 The Mediator/Wrapper Approach to Information Integration

In a mediator/wrapper architecture (Figure 9) for information integration each data source is "wrapped" and the mediator provides a uniform view of the data.

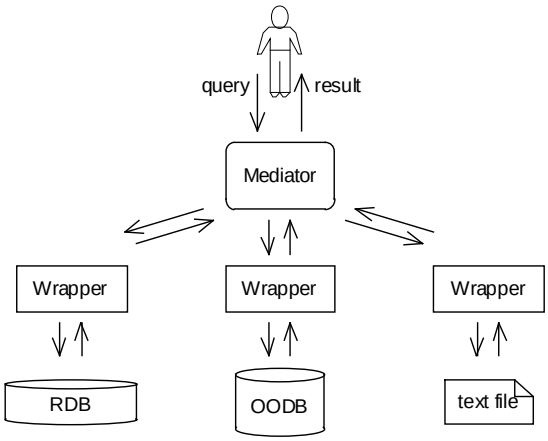


Figure 9. Mediator/wrapper architecture (after Garcia-Molina et al 2000).

In the following sections wrappers and mediators will be described separately. Two things from section 4.3 to bear in mind are that most data sources are fully autonomous (out of control of the administrator of the mediator/wrapper system), and that the data sources may be highly heterogeneous (different data models, database schemas, and querying capabilities).

The Amos II system used to implement the prototype of this Thesis has a mediator/wrapper architecture with a functional OO data model[81]. Other research mediator/wrapper-systems are Garlic[82] (from IBM), Tsimmis[83] (Stanford & IBM), Information Manifold[84] (AT&T), and Disco[85] (INRIA).

[77] When we deal with experimental data there is often no right-up "correct" value. E.g. we may find that different databases store widely different values of the affinity constant K_{AB} between binding site A and ligand B, and this may be due to the different experimental conditions used. I.e., we can normally not *assume* that two data values of K_{AB} refer to exactly the same real-world concept.

[78] E.g., `Employee` of schema A is certainly a subtype of `Person` in schema B.

[79] E.g., it is possible to derive the value of `molecularWeight` in schema A from values of `molecular-Composition` and `atomicWeight` in schema B.

[80] Boman, Bubenko, Johannesson and Wangler 1997.

[81] Risch, Josifovski and Katchaounov 2001.

[82] Tork Roth and Schwarz 1997; Haas, Miller, Niswonger, Turk Roth, Schwarz and Wimmers 1999.

IBM's DiscoveryLink (Haas et al 2001, section 6.2.1) is a system with purpose and architecture very similar to PAQS. DiscoveryLink is based on the wrapper/mediator architecture of Garlic, but uses IBM's ORDBMS DB2 UDB as integrating middleware.

Typically, a mediator is constructed for a particular group of users, and the data sources might then all contain "similar" data - from a single information domain. One advantage of this specialisation is that each "local" mediator with associated data sources can be used as a component in a larger, extensible, distributed system (Özsu and Valduriez 1999, p 586). This would be possible with, e.g., Amos II mediators as components (Risch and Josifovski 2001). Figure 10 depicts such a system of collaborating or co-operating mediators.

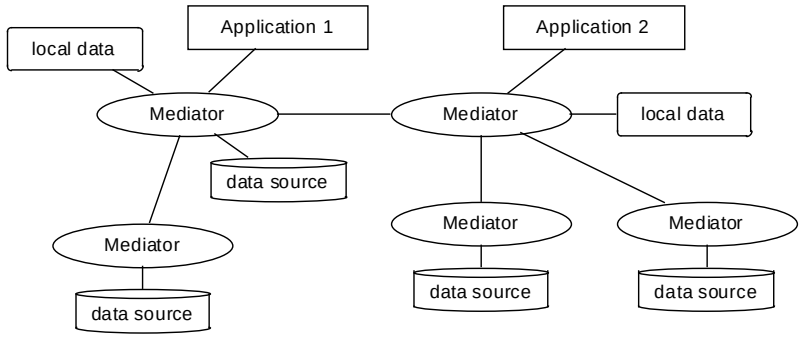


Figure 10. A system of collaborating mediators (after Josifovski 1999). Mediators with the sole responsibility to wrap a data source (the three at the bottom) may be called "translators" (Josifovski 1999).

A new part of SQL, called Management of External Data (or SQL/MED, see section 4.4.3), can be used to access external data through wrapper interfaces (Melton, Michels, Josifovski, Kulkarni, Schwarz, and Zeidenstein 2001).

4.4.1 Mediators

In previous sections we have defined a mediator as a software component which provides a uniform view of data residing in various (heterogeneous and autonomous) data sources. Wiederhold (1992) emphasises that mediators should use domain knowledge as a driving force for data integration, and that the purpose of the mediator should be to create information which can be used in high-level (decision-making) applications.

Wiederhold and Genesereth (1997) describe mediation as an architectural concept. They give three layers:

- A foundation layer with base resources (databases and simulation programs).
- A mediation layer with value-added services and domain-specific code.
- An application layer where decision-makers use the information.

An important point is that while the foundation layer is managed by database administrators, the mediation layer should be managed by domain specialists (Wiederhold 1992).

[83] Hammer, Garcia-Molina, Ireland, Papakonstantinou, Ullman and Widom 1995; Hammer, Garcia-Molina, Nestorov, Yerneni, Breunig and Vassalos 1997.

[84] Levy, Rajaraman and Ordille 1996.

[85] Tomasic, Raschid and Valduriez 1998.

According to Wiederhold and Genesereth (1997) a mediator needs to accomplish four main tasks in order to fulfil the overall objective of adding value. The mediator should be able to

- access and retrieve data from multiple heterogeneous sources,
- abstract and transform data into a common data model and a common database schema,
- integrate the homogenised data, and
- reduce the integrated data by abstraction in order to increase the "information density".

As discussed in the next section, much of this functionality can be accomplished by means of cleverly designed wrappers.

4.4.2 Wrappers

A mediator may communicate with a data source through a wrapper (see Figure 9), which describes the data in the source and also provides a mechanism by which the mediator can retrieve the data. Basically, we may see the wrapper as a "simulation layer" on top of the source[86].

In a mediator/wrapper architecture there are several things a wrapper should be capable of. Obviously, the wrapper should model the contents of the data source in the common data model of the mediator. This corresponds to presenting an export schema in Figure 8.

Secondly, a wrapper should allow the mediator to access data from the source. In order to do this, the wrapper must be able to translate mediator queries (often an object version of SQL) into whatever access methods the source has (SQL, web forms, open a file and read sequentially, et cetera).

In addition to these two basic tasks some mediators require the wrapper to participate in query planning. I.e., the mediator should be able ask the wrapper for how much of a query it can answer, and at which cost. This is the approach taken by Garlic (Tork Roth and Schwartz 1997) and DiscoveryLink (Haas et al 2001).

4.4.3 SQL/MED

SQL/MED (Management of External Data) is a new part of SQL that describes how standard SQL can be used to concurrently access external SQL and non-SQL data (Melton et al 2001). The architecture of SQL/MED is shown in Figure 11.

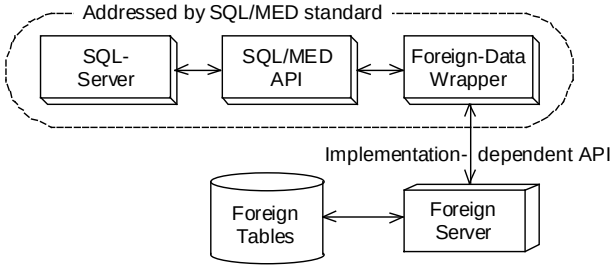


Figure 11. Components used by SQL/MED (Management of External Data) (after Melton et al 2001).

[86] This is the view of Stonebraker and Brown (1999) when they describe how a wrapper can be used to extend a traditional relational DBMS with object-relational functionality.

The new SQL/MED standards seems to be strongly influenced by IBM's Garlic project. In the SQL/MED approach, the SQL database server breaks a query into fragments which are to be processed by different foreign servers. In the query planning phase each query fragment is converted to an SQL/MED request and sent to a wrapper, which analyses the request and constructs an SQL/MED reply together with an execution plan. The server then investigates the replies of the sources, and if it finds that the sources together are capable of answering the original query it constructs an overall execution plan.

In the query execution phase the SQL database server sends the partial execution plans back to the appropriate wrappers. Each wrapper then accesses its external tables according to its execution plan, and returns the results to the SQL-server, which finally constructs an overall answer to the query.

The present SQL/MED standard (called MED:2000) only allows simple queries and a read-only interface to external data. The next version (due late 2002 or early 2003) is expected to allow for modifying data in external tables (Melton et al 2001).

5 Life Science Data Sources and Formats

During the last 20 or so years there has been an increasing interest in applying databases to biological data. Most research has been concerned with genetic codes, amino acid sequences of proteins (1º protein structure), and 3-D protein structures (3º and 4º structures). This shows, for instance, when examples of biology data are given in texts for the database community[87].

In this chapter available life science data formats and data sources on the Web will be discussed. Then, in the next chapter (6) a few database systems and standardisation projects for integration of biological data will be described. Since the number of data sources and data formats is large[88] and rapidly expanding I will only describe a selection of them. I have selected those sources and formats I believe are most useful or interesting for the PAQS project.

5.1 Requirements on a Web Data Source

Markowitz, Chen, Kosky, and Szeto (2001) discuss a few criteria for how molecular biology databases on the Web can be evaluated and compared. Since their survey of some major web data sources was made 1996, details will have changed, and I will not discuss the individual data sources. However, a few general points of interest, and of relevance for the PAQS project, are metadata availability and query capabilities.

Markowitz et al point out that in order for a web database to be really useful to the public it should provide comprehensive on-line metadata. For example, it is essential that users can see which entity types and relationship types the schema consists of, and which constraints that apply on the data. However, many web data bases had (1996) little or no metadata on-line, which according to Markowitz et al probably was due to the fact that schemas are revised frequently and that it is laborious to keep such information up-to date on a web site.

It was further pointed out that *on-line metadata browsing* is a very useful facility. This is something which should be easy to provide with Amos II since all metadata are stored as objects in the database. Actually, the program goovi (Cassel and Risch 2001) presently functions as a combined data and metadata browser for Amos II databases. What is needed is to make this kind of information available to web browsers and, possibly, to restrict the browsing to metadata.

Markowitz et al (2001) further discuss different ways to query databases. They suggest that a web site should provide fixed-form queries for the most common query types, as well as support for *ad hoc* queries, e.g. in SQL.

5.2 Data Formats for Information Exchange in Bioinformatics

Of special interest in the networked modern world is how an application can get hold of data from different data sources. This "information integration" aspect was discussed upon in chapter 4. From a practical point of view, information integration is easier the more standardised the data formats of the various data sources are, and in this section we will

[87] Paton 2001; Bellahsene and Ripoche 2001; Hammer and McLeod 2001.

[88] The DBCAT catalog currently lists 511 databases in molecular biology, covering DNA, protein structures, literature, et cetera (http://www.infobiogen.fr/services/dbcat/, 2002-01-23).

describe a few data formats that are in use for genomes, proteomes and other biological data, and also a few that have recently been proposed.

5.2.1 Flat File Data Formats

Several large web databases present users with the data on "flat file" formats. Such data typically consist of text on a very strict format, suitable for old FORTRAN-style I/O. These data sources existed before the birth of the Web, and data can in many cases still be downloaded by ftp.

As we will see in the following sections several large data sources have gone over to managing their data with modern DBMSs. For backward compatibility users are still presented with the "flat file" data, but some servers have in parallel started exporting data as "semi-structured" XML files, or as CORBA/Java objects.

5.2.2 ASN.1 at NCBI

ASN.1 (Abstract Syntax Notation One)[89] is a formal notation for describing data transmitted by telecommunications protocols. The notation contains basic types such as integers, character strings, and booleans (but not reals). There are also more complex types, e.g. structures, lists, and choices.

A specification written in ASN.1 is similar to an XML DTD, and we can treat such a specification as a schema over the domain of interest. However, ASN.1 is not a data model as it lacks support for integrity constraints and data manipulations. Since data in ASN.1 documents are "tagged", we say that the format is *self-describing*.

Although ASN.1 was originally designed as a part of the OSI (Open System Interconnection) standard in telecommunications it has spread into other areas, e.g. biology. One a strong point of ASN.1 is the associated encoding rules for effectively transforming a data file to signals that can be transmitted. Of more interest in the web age is XER, encoding rules for transforming between ASN.1 and XML[90].

NCBI (the National Center for Biotechnology Information) uses ASN.1 as specification language for schemas. Although it is possible for a human to read small ASN.1 documents, larger ones get messy. At NCBI most data is presented in users in flat file formats (see, e.g. GenBank, section 5.5.2). There are publicly available programs for converting from ASN.1 to, e.g., GenBank "flatfile format", and even a C library[91] for handling ASN.1 files. With help of XER NCBI has recently started publishing data in XML format.

5.2.3 XML

XML, the eXtensible Markup Language, provides a standard way of defining a set of tags (a "vocabulary") for a domain of interest. Such a set of tag definitions, possibly together with the relationships between them, constitutes a new markup language - designed for the specific domain of interest. I.e., XML is used to define which tags that can be used in the new

[89] *ASN.1 Information Site* (2001-12-12).

[90] Larmouth 2001.

[91] *NCBI toolbox* (2001-12-12).

language, and then this new language is used for documents or data. Since XML describes the structure of data it could be used for defining the structure of heterogeneous databases and data sources[92,93].

In the following subsections a few XML-related topics will be described very briefly. For more information, see, e.g., the official XML home page (http://www.w3.org/XML/). In section 5.2.4 various custom markup languages for life science applications will be described.

DTDs and XML Schemas

The technical definitions in the new language are captured in a Document Type Definition (DTD) or in an XML Schema. DTDs are the older mechanism, and more validating parsers are available for the DTD format. However, the XML Schema standard is more expressive. XML schemas are first class XML documents, and hence schemas and documents can be edited and processed by the same tools. Thus, I believe that XML Schema will be the more used mechanism as the number of parsers increases.

XML APIs

DOM and SAX are the two widely used standards for parsing of XML. DOM (the Document Object Model) is a tree-based API which provides an object-oriented view of the data. A DOM parser transforms the complete XML document to a tree structure in main memory, and this tree can then be traversed, queried, and manipulated. Conversely, SAX (the Simple API for XML) is event-based and provides serial access to the data. The XML document is parsed and appropriate events are fired, but no main-memory data structure is created.

JAXB (Java XML Binding) is a third, non-generic, approach for binding Java applications to XML data. It is based on the idea that a specific XML schema (or DTD) can be directly represented as a number of Java classes. Thus, JAXB consists of a compiler which generates Java classes from an XML Schema (or DTD), and a runtime framework which transforms an XML instance into Java objects and vice versa[94].

RDF and Interoperability

Although XML Schema provides a method for defining the structure of XML documents within a domain it does not support interoperability[92]. There are several ways to build XML schemas (or DTDs) for a domain. Thus, laborious mappings and translations might be needed before two application which use different schemas can exchange data.

The Resource Description Framework (RDF) is an infrastructure that enables the encoding, exchange, and reuse of metadata[92]. It is the basis of W3C's *semantic web* and "the RDF specifications provide a lightweight ontology system to support the exchange of knowledge on the Web" [95]. More informally, with RDF you can "use any metadata you like, but make the description of them publicly available" [96]. I am not aware of any use of RDF for information integration in the life sciences.

[92] Connolly and Begg 2002, ch 29.

[93] A useful technique in this context could be XSLT (eXtensible Stylesheet Language for Transformations) which can be used to transform an XML structure into, e.g., another XML structure, HTML, or SQL.

[94] *Java Architecture for XML Binding* (2002-01-17); Thelen 2001.

[95] *Resource Description Framework (RDF)*, http://www.w3.org/RDF/ (2001-12-17).

[96] Rehn 2000.

XMI

The XML Metadata Interchange Format (XMI) specifies an open information interchange model that is intended to give developers working with object technology the ability to exchange programming data over the Internet in a standardised way[97].

The only use of XMI for life science data management that I am aware of is the creation of a custom markup language (MAGE-ML) from an object model in the MAGE project (see section 5.2.4). Thus, in this case XMI was not used for exchanging schemas between applications, but only for creating a DTD from a schema.

5.2.4 Custom Markup Languages

XML is a metalanguage, a language for creating other languages. This is why XML is interesting in the context of the PAQS project. Various organisations and companies have produced their customised version of XML, i.e. custom markup languages. Most such efforts have simply defined a DTD (document type definition) and then tried to convince others to use it. An alternative, not yet explored by many, is to create an XML Schema over the domain of interest.

It might be useful to differ between such markup languages that are created through collaborations between several companies or laboratories, possibly within some recognised body of standardisation, and those languages which are created by a single data provider due to client requests or supposed marketing advantages. We might suppose that a markup language of the later kind will closely follow the way clients use the original data source, and that it is probably not as general as the former kind of language. However, from the point of immediate usefulness the "single-data-source" language might be better, since the data should already be available.

As far as I have found, no organisation has tried to sell its DTD. Instead, each believe it to be a competitive advantage to get as many users as possible, and DTDs are for free. In addition, many organisations and companies give away executables and source code for browsers and application programs associated with the customised markup language.

In the following subsections a few custom markup languages will be described and discussed. The first is CML, mainly for small molecules and crystal structures. Then I have chosen BIOML, AGAVE, BSML, and PROXIML as examples of markup languages for genomes and proteomes. I believe that BSML is the one which eventually "win the race", i.e. get most followers over the next year or so. The last, MAML and MAGE-ML are chosen as examples of markup language for a domain with large volumes of experimental data.

Other languages which are not directly suited for the PAQS project could still provide ideas. Two such languages are SBML and XSIL[98].

To my knowledge no custom (XML) language presently exists for the representation of binding assays and binding experiments. Work is underway to construct a DTD for BIND (section 5.4.1). The development of XML Schemas for binding experiments and similar topics could clearly be a useful part of the PAQS project.

[97] *XML Metadata Interchange Format (XMI)*, http://www-4.ibm.com/software/ad/library/standards/xmi.html, (2001-12-16).

[98] SBML (the Systems Biology Markup Language), http://www.cds.caltech.edu/erato/sbml-level-1/sbml.html (2001-02-19); XSIL (the eXtensible Scientific Interchange Language), http://www.cacr.caltech.edu/XSIL (2001-02-19).

CML - The Chemical Markup Language

The Chemical Markup Language (CML) was designed as an "HTML for molecules" [99]. CML concentrates on "molecules" (discrete entities representable by a formula and usually a connection table), and supports reactions, compound molecules (clathrates, macromolecules, etc.), and macromolecular structures/sequences. However, CML has no specific support for physicochemical concepts, but allows quantities and properties to be specifically attached to molecules, atoms or bonds.

According to the creators has CML been used together with XHTML (for text and images), SVG (line diagrams et c.), PlotML (graphs), MathML (equations), XLink (hypermedia), RDF and Dublin Core (metadata), and XML Schemas (for numeric and other data types). DTD and XSD (schema) files are available from the official web page (http://www.xml-cml.org/). A range of useful resources are available from CML's homepage, e.g. a browser and CML-DOM parser.

It seems that CML has a strong focus on "publications", e.g. how web and print documents can display molecules and related information transparently. Thus, from the viewpoint of CML it is an advantage that XML supports "documents" and "data" in a seamless spectrum. In my opinion, CML should be extended (or combined with some other markup language) in order to provide a schema for experimental data of all kind.

Murray-Rust started the development of CML very early, and CML is (together with MathML) often taken as an example of a custom markup language[100]. Although a white paper[101] on CML claims that many companies and organisations have adopted CML, few examples are given.

BIOML

BIOML[102] seems to have been one of the first markup languages for proteins and genes. It was produced by the company Proteomics (http://www.proteomics.com; http://www.bioml.com) for specification of biopolymer sequences, experimental information, and annotations. Proteomics explicitly mentions that the markup language should facilitate exchange of structure and annotation information over the Web.

A BIOML document describes a physical object, e.g., a particular protein, in such a way that all known experimental information about that object can be associated with the object in a logical and meaningful way. Such structures can then be viewed with Proteomics' free BIOML browser (which uses a quite imaginative and figurative typefont for different letters).

The DTD is publicly available, but the project seems to have come to an end in 1999.

BSML

The Bioinformatic Sequence Markup Language (BSML)[103] is, according to its creators LabBook, Inc (http://www.labbook.com), a "public domain standard" for the encoding and

[99] http://ww.xml.cml.org, *Chemical Markup Language* (2001-12-14); Murray-Rust, Rzepa, Wright and Zara 2000.

[100] For example, in Deitel, H.M., Deitel, P.J., Nieto, T.R., Lin, T.M. & Sadhu, P. (2001) *XML - How to Program*, Upper Saddle River, N.J.: Prentice Hall as well as in Åström, P. (1999) *XML, Extensible Markup Language* (in Swedish), Stockholm, Sweden: Docendo.

[101] Murray-Rust and Rzepa 2001.

[102] *The BIOML home page*, http://www.bioml.com/BIOML/index.html (2001-12-14); Fenyö 1999.

[103] http://www.bsml.org/, *BSML - An Emerging Industry Standard* (2001-12-14).

display of DNA, RNA and protein sequence information. This markup language has gained dramatically in support over the last half-year. Since the creation of I3C (the Interoperable Informatics Infrastructure Consortium, section 5.3.4), where LabBook is a founding member, organisations such as IBM, Bristol-Myers Squibb, and EBI (European Bioinformatics Institute) have created alliances with LabBook and/or stated support for BSML[104]. LabBook also provides a free Genomic XML Viewer and sells a Genomic XML Browser.

A distinguishing feature of BSML is that documents consist of two parts, a "definitions section" with bioinformatics data, and a "display section" with information on how the data should be represented graphically. Thus, the publicly available DTD is quite long, and the second section contains, e.g., elements concerning paper margins, the plotting of pie-charts, and the display of electrophoresis gels. These are thing we really do *not* want in a language for data exchange, they are instead related to publication of the data on the Web. However, this section is optional, and a DTD for data only is also available.

If the I3C (or a few of the other major collaborations of LabBook) prove successful, then BSML will no doubt become the dominant markup language for exchange of biopolymer structures and annotations. However, the language is still evolving, and LabBook welcomes comments and suggestions on the present DTD (version 2.2).

AGAVE

AGAVE (Architecture for Genomic Annotation, Visualization and Exchange)[105] is an XML format developed by DoubleTwist, Inc. (http://www.doubletwist.com) for managing, visualising and sharing annotations of genomic sequences. As in BIOML, the available DTD contains no elements referring to graphical representation, this is handled by DoubleTwist's Genomic Viewer.

AGAVE is perhaps mainly intended to be used together with Prophecy, DoubleTwist's Annotated Human Genome Database system, which includes an AGAVE compliant XML repository of the genomic annotations and uses AGAVE as a data exchange medium. However, currently EBI (the European Bioinformatics Institute, vide supra) provides its EMBL/GenBank/DDBJ data in two XML formats only, apart from the traditional "flatfile" format, and these are BSML and AGAVE.

PROXIML

PROXIML (the PROtein eXtensIble Markup Language)[106] is the only biopolymer markup language I have found which utilises XML Schema instead of DTDs. PROXIML builds on CML, but is most likely not used by any of the major databases or integration projects.

MAML and MAGE-ML

The MicroArray Markup Language (MAML) is an effort of the MGED (MicroArray Gene Expression Database) group to design a standardised markup language for the domain of microarray gene expression experiments[107].

MAML, and any effort to standardise microarray data, needs to deal with the following five subdomains[108]: (i) the array, (ii) sample sources, treatments, and samples, (iii) hybridisations,

[104] See http://www.bsml.org/news for a full list of press releases.

[105] http://www.agavexml.org/, *Welcome to AGAVE* (2001-12-14).

[106] McArthur 2001, http://www.cse.ucsc.edu/~douglas/proximl/.

[107] Cover 2001; *MGED group* (2001-12-17), http://www.mged.org/.

[108] *Array XML (AXML)*, http://www.xml.org/xml/zapthink/std409.html (2001-12-17).

(iv) data, and (v) analysis. These subdomains are similar to what is needed for the description of binding assay experiments, while the details will of course differ.

One important problem in storing or exchanging microarray data in some XML format is how large amounts of numerical data should be represented[109]. In MAML this is solved by storing binary or ASCII data on an external file (specified by an URI (uniform resource identifier)). It is also possible to store (untagged) ASCII data within an XML document.

The MicroArray Gene Expression Markup Language (MAGE-ML)[110] is part of a larger project, where an object model (MAGE-OM) for gene expression data is defined according to OMG's Model Driven Approach (MDA). In MAGE-ML too, it is foreseen that the XML document shall reference external data files. By inspection of the MAGE-ML DTD it seems that data are organised in three-dimensional "cubes", with dimensions "BioAssay", "DesignElement", and "QuantitationType". I have, however, not tried to delve into the details of the MAGE object model or markup language, and they have not influenced the schemas presented in chapter 8 of the Thesis.

An interesting difference between MAML/MAGE-ML and the previously described custom markup languages is that MAML and MAGE-ML are standardisation efforts, made by several co-operating laboratories. In the case of MAGE-ML one even uses the strict procedures of OMG.

5.2.5 mmCIF

CIF (the Crystallographic Information File) is a format for describing crystallographic experiments and the structures of small organic molecules. CIF was developed by the International Union of Crystallography (IUCr) and is the preferred format for submitting data to the various IUCr journals. Hence, mmCIF has the potential to be an important exchange format in all areas of chemistry.

mmCIF is a CIF dictionary specific for biological macromolecules (hence *mm*)[111]. mmCIF contains over 1700 terms and provides the conceptual schema for the new Protein Data Bank of RCSB (section 5.5.4). A macromolecular structure in mmCIF format consists of name-value pairs, with a loop construct for repeating record types (e.g. the atomic co-ordinates for each atom). It is possible to convert from mmCIF format to the more familiar PDB format, but the opposite is not always true. (The PDB format is too informal.)

Westbrook and Bourne (2000) give a larger picture of STAR/mmCIF as an ontology for macromolecular structures. STAR (Self-defining Text Archival and Retrieval) has been used to define a data dictionary language (DDL). This DDL provides conventions for naming and defining data items within a dictionary, and at least eight dictionaries have been built - one of which is mmCIF. Finally, the data resides in data files which conform to the dictionary. The data files, with name-value pairs, constitute a form of semistructured data, and it should be easy to convert mmCIF into an XML DTD.

---

[109] From reference 108: "Imagine a matrix of 10,000 genes by 1000 arrays. If each element-hybridization pair resulted in 20 numbers (each needing 20 bytes for XML encoding) this document would be 400 x 1,000 x 10,000 = 40 GB for the complete file. Clearly a more compact representation must be allowed."

[110] http://www.geml.org/omg.htm

[111] Bourne, Berman, McMahon, Watenpaugh, Westbrook and Fitzgerald 1997.

5.3 Standardised Protocols for Information Exchange

In this section three different projects will be described, which all aim at producing standard protocols for exchange of life science data between applications. The first two use CORBA, and the third use XML.

5.3.1 CORBA

CORBA (Common Object Request Broker Architecture) is a standard for the exchange of objects between applications and common object services[112]. In a CORBA environment objects have interfaces defined in IDL (Interface Definition Language). Through these interfaces requests and messages can be passed between objects distributed over a network. A major advantage of CORBA is that the objects can be living in applications written in *different* object-oriented languages. Thus, distributed objects can co-operate irrespective of programming language, hardware, operating system, or geographic location, as long as they adhere to the same IDL interfaces.

Obviously, these IDL definitions need to be general and stable. Hence, one way of providing for interoperability between applications within a specific domain would be to develop a standardised set of IDL interfaces for the domain. Within molecular biology/bioinformatics there are at least two such standardisation efforts (vide infra).

Note that in contrast to ODMG's object data standard (see section 3.2.3) OMG's CORBA is not specifically intended for database systems. However, as long as the DBS presents an appropriate IDL interfaces for its types CORBA could be used to access it. For example, the P/FDM system has been given a CORBA interface (see section 6.1.1).

It should also be noted that interest in CORBA has recently decreased due to the appearance of XML as a possible format for data exchange[113].

5.3.2 The Life Sciences Research group

The Life Sciences Research (LSR) group[114] is a consortium of pharmaceutical companies, software and hardware vendors, and academic institutions working within the Object Management Group (OMG) in order to improve interoperability among object-oriented life science research applications. Examples of life science research areas covered by the LSR group are bioinformatics, genomics, cheminformatics, computational chemistry, and clinical trials. The LSR group works not only with CORBA, but also with other (object) technologies such as UML, XML, and EJB (Enterprise Java Beans).

The LSR group is formally a Domain Task Force (DTF) of OMG, and it follows OMG's process to standardise models and interfaces. The work is performed by ten work groups (WG). Three work groups of particular interest to the PAQS project are (i) the Chemiformatics WG (CORBA interfaces for drug discovery research), (ii) the LECIS WG (laboratory equipment control interface specification), and (iii) the Workflow WG (to

[112] CORBA is developed by OMG (Object Management Group), a consortium of major software vendors (http://www.omg.org). A starting point for learning about CORBA is a *CORBA FAQ* page provided by OMG: http://www.omg.org/gettingstarted/corbafaq.htm (2001-12-07).

[113] Davidson, Crabtree, Brunk, Schug, Tannen, Overton and Stoeckert 2001.

[114] *Introduction to LSR* (2001-12-06).

describe and control flow of data between components, e.g. to represent scientific experiments). In addition, there are work groups on themes such as 3-D macromolecular structures, sequence analysis, and microarray gene expression data.

5.3.3 The EU-funded CORBA project

The European Union funds a research project[115] which aims to combine the data and services of a number of European biological databases using CORBA. One of the project partners is EBI (the European Bioinformatics Institute) where the EMBL Nucleotide Sequence Database has been made available through CORBA servers[116]. Since the original EMBL data are published on flat file format this clearly has the potential of increasing the availability for OO applications.

EBI is one of the major stakeholders in the LSR group too, and there is some work to make the EU project's IDL comply to that of the LSR (or the other way around). Presently, the two standardisation efforts have not produced compatible interfaces, however.

5.3.4 The Interoperable Informatics Infrastructure Consortium

The Interoperable Informatics Infrastructure Consortium (I3C)[117] develops common protocols and interoperable technologies (specifications and guidelines) for data exchange and knowledge management for the life science community. In contrast to other standardisation efforts (e.g. the LSR-DTF in section 5.3.2) I3C will also develop technology solutions, not only standards.

I3C was created in January 2001 and has over 60 participating life science and information technology organisations[118]. The goals of this relatively new organisation are (i) to facilitate an open development of standards, protocols, administration and a technical infrastructure for the life science industry, (ii) to establish a common communications standard protocol that is extensible and can be delivered to the community in a timely fashion, and (iii) to provide forums for discussion of issues that affect technology evolution, development and use.

The main product of I3C so far seems to have been a demonstration[119] of a working prototype at the BIO2001 conference[120]. The demonstration was meant to represent a workflow of the typical molecular biologist who identifies a collection of sequences of interest and performs a series of analyses on those sequences to further explore the data (see Figure 12). The prototype relies on Java and *an open XML-in, XML-out paradigm*, and used BSML of LabBook (section 5.2.4) as communication protocol. What is rather impressive is the integration of

[115] Contract: BIO4-CT96-0346. *"Linking Biological Databases …"* (2001-12-06).

[116] *CORBA at EBI* (2001-12-06).

[117] *I3C Home Page*, http://www.i3c.org/ (2001-12-14).

[118] I3C was originally indented for industrial partners only, but now has also governmental agencies and universities as members. For a list of participating organisations (as of July 28th, 2001), see http://www.i3c.org/html/i3c_faq_june01.htm (2001-12-14).

[119] Unfortunately, there is no publicly available web site where one can test the concepts. For those of us who were not present at BIO2001 the only available information is a single web page: *I3C Demo at BIO2001*, http://www.i3c.org/Bio2001/i3c_demo_final.htm (2001-12-14).

[120] BIO 2001 is an international convention and exhibition organised by the Biotechnology Industry Organization in San Diego, June 2001, see http://www.bio.org/events/2001/event2001home.html (2001-12-19).

technology from eight different companies and organisations[121] for this demonstration. This, I believe, shows the potential strength of I3C.

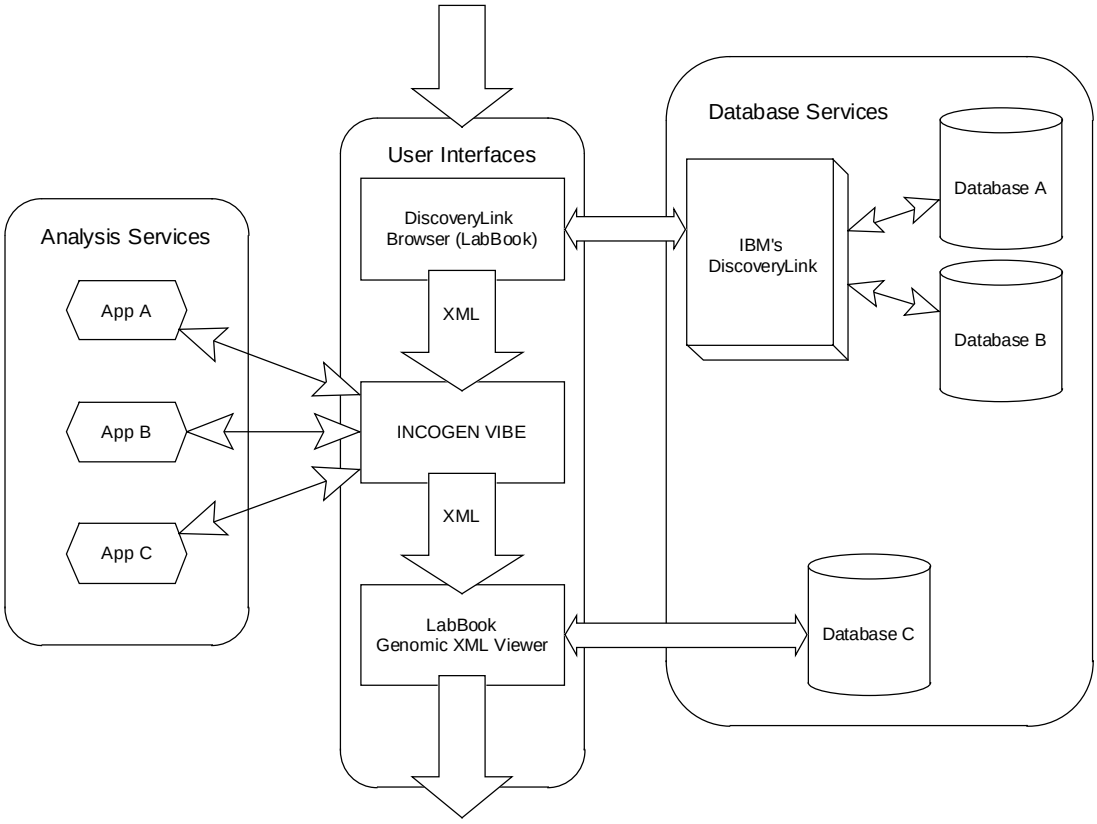


Figure 12. Architecture of the I3C demonstration prototype. (After *I3C Demo at BIO2001*.)

Of particular interest is that IBM's DiscoveryLink (section 6.2.1) and LabBook's BSML standard (section 5.2.4) play central roles in the I3C architecture.

As seen in Figure 12 the product VIBE from INCOGEN[122] has a central position in the I3C demonstration set-up. VIBE (Visual Integrated Bioinformatics Environment)[123] is a tool for visual data analysis and mining. In VIBE a user may construct a "pipeline" of data filtering and analysis modules by means of connected icons on the screen. This pipeline can then be saved as XML or executable Perl code. According to the information available on the web site, VIBE is designed specifically to work with analysis programs of the Decypher platform from TimeLogic[124]. However, VIBE is extensible, and this has obviously been utilised in the I3C demo.

[121] Blackstone, IBM, INCOGEN, LabBook, National Cancer Institute (NCI), Sun, TimeLogic, and TurboGenomics.

[122] The Institute for Computational Genomics, Inc., http://www.incogen.com (2002-01-10).

[123] http://www.incogen.com/vibe/; *INCOGEN VIBE* (2002-01-10).

[124] http://www.timelogic.com/biodata.htm (2002-01-10).

5.3.5 The ISYS platform

The ISYS platform[125] is a means for application interoperability which does *not* depend on a standardised protocol. It cannot even be said to be a standardisation effort, since it emanates from a single laboratory[126]. However, it is presented as an alternative to the CORBA approach (sections 5.3.1-5.3.3), and as such it is interesting to mention in this section. The architecture is similar to CORBA, with software components registered as service providers and/or event listeners.

What distinguishes ISYS from other solutions is a process called "interactive discovery", where registered components at run-time are interrogated whether they can operate on some data selected by the user. Those components which *can* operate on the data are added to a menu in the GUI, together with appropriate descriptions. Thus, there is no common schema or interface but instead each "data producer" can give an object as many or few attributes as desired, and different "data consumers" may have different views of the same object. The system is implemented in Java, and existing databases and applications are integrated by wrapping with Java classes.

The authors conclude that the design makes ISYS suitable for systems where there is little overlap between the data used by different components, and that the solution will be ineffective if various components should share large parts of their schemas.

5.4 Binding Affinity Data Sources on the Web

The journal Nucleic Acid Research yearly publishes a "molecular biology database collection" (Baxevanis 2001)[127]. A few of these data sources provide binding affinity data.

Note that some of the projects described in this section may very well be thought of as projects for information integration, and could have been put in chapter 6. However, I have tried to keep sources which focus on a single kind of data in this section, and reserved chapter 6 for projects with a wider scope.

5.4.1 BIND - the Biomolecular Interaction Network Database

The Biomolecular Interaction Network Database (BIND) is a database of interactions between pairs of objects[128]. Each such interacting object may be a protein, DNA, RNA, ligand, molecular complex, or other biochemical entities. The scope of BIND is rather wide[129]. The database contains information about experimental conditions used to observe the interaction, cellular location, kinetics, thermodynamics et cetera. To cite the web page[130]:

> "Development of the BIND 2.0 data model has led to the incorporation of virtually all components of molecular mechanisms including interactions between any two molecules composed of proteins,

[125] Siepel, Tolopko, Farmer, Steadman, Schilkey, Perry and Beavis 2001.

[126] NCGR, the National Center for Genome Resources, Santa Fé, New Mexico.

[127] The list from January 2001, with accompanying short papers by the database providers, is publicly available at http://nar.oupjournals.org/content/vol29/issue1/ (2001-12-09).

[128] Bader and Hogue 2000; Bader, Donaldson, Wolting, Ouellette, Pawson, and Hogue 2001.

[129] BIND could very well have been placed as a warehousing integration project in section 6.3.

[130] *BIND - The Biomolecular Interaction Network Database* (2001-12-10).

> nucleic acids and small molecules. Chemical reactions, photochemical activation and conformational changes can also be described. Everything from small molecule biochemistry to signal transduction is abstracted in such a way that graph theory methods may be applied for data mining. The database can be used to study networks of interactions, to map pathways across taxonomic branches and to generate information for kinetic simulations. BIND anticipates the coming large influx of interaction information from high-throughput proteomics efforts including detailed information about post-translational modifications from mass spectrometry."

The data format used by BIND is ASN.1 (Abstract Syntax Notation.1, section 5.2.2), the reason being that NCBI (the US National Center for Biotechnology Information) uses ASN.1 to describe and store its biological and publication data. BIND also uses the NCBI database schema[131], extended with many additional types and attributes.

We will not here go into the details of the fairly complex BIND schema. It is available as an ASN.1 specification[132] from ftp://ftp.bind.ca/BIND (where also the database files and some source code is available). The schema is also described by Bader and Hogue (2000), in text as well as in (slightly incorrect) UML diagrams.

Three central concepts are BIND-interaction, BIND-object (two for each interaction) and BIND-desc (descriptions of interactions). Each object has an "object type id", which is a choice[133] between protein, ligand, DNA, et cetera. Further each object has an "object origin" (choice between organism, chemical, and not specified). It is obvious that UML is not suited to graphically represent the semistructured ASN.1 data format. Unfortunately, neither the web site nor the publications use some form of tree to graphically describe the schema.

The creators of BIND have made a point out of using the data format and schema of NCBI. They note that the schema is "mature", i.e. that the core of it is stable, and does not change as often as many other schemas. Furthermore, as pointed out by Ostell, Wheelan and Kans (2001) NCBI is a US Government agency, with the possibility to play a long-term role in bioinformatics.

BIND can be queried through a web form, but this functionality seems fairly poorly developed. The public web form for querying BIND is not very detailed. For example, in the "advanced query" form one can only choose between a text search and a search on BIND accession ID numbers.

It is also relevant to consider how BIND treats non-public data. In general, a BIND object which describes a biopolymer sequence will store a link to a sequence database, e.g. GenBank. Sequences which are not publicly available may be represented by a NCBI-Bioseq object. Further, each interaction may be labelled private, in which case the record is not exported during data exchange, but may be viewed in-house.

The database is growing[134], and new data may be submitted through web forms. New data is indexed by BIND staff and validated by a scientist. This ought to ensure good quality of the data.

The BIND project originated in academia, but some of the people in the project are also affiliated with MDS Proteomics, Inc (see the following subsection).

[131] In BIND and NCBI publications this is called the NCBI data *model*. In this Thesis I try to make a distinction between schema and model.

[132] Work is also underway to define an XML DTD translation from ASN.1.

[133] CHOICE is a construct of ASN.1. It is in essence an enumerated type, only more complex. Perhaps the best way to represent a CHOICE in UML would be as disjoint subclasses of an abstract class.

[134] March 2001: 5805 interactions, Dec 2001: 5939 interactions.

MDS Proteomics, IBM, and BIND

In January 2001, MDS Proteomics[135] and IBM announced having formed a "strategic alliance". MDS Proteomics will start using IBM's DiscoveryLink 6.2.1 as a data integration technology, and IBM will invest in MDS Proteomics.

The really interesting point, however, is that IBM and MDS Proteomics together will work to establish the database BIND. In the press release BIND is characterised as "a publicly available bioinformatics database that will allow researchers world-wide to submit and review results of research about molecular interactions and the detailed cellular mechanisms of life" and the two companies make "a long-term commitment to continued support" [136].

If the commitments expressed in the press release are fulfilled we can expect BIND to have a strong position as a data source in bioinformatics.

5.4.2 DIP – Database of Interacting Proteins

DIP (Database of Interacting Proteins)[137] is a database of protein pairs that are known to interact with each other. DIP is publicly available through a web form. The service is intended to aid scientists who study protein-protein interactions, signalling pathways, multiple interactions and complex systems. New data can be entered through a web form, but is curated before it is made publicly available.

DIP is implemented as three tables in a relational mySQL database: Proteins, Interactions, and Experiments. Proteins can be searched by PIR, SWISS-PROT, or GenBank identification codes. The database contains information (if available) about the protein regions involved in the interaction, the dissociation constant and the experimental methods used to study the interaction.

Data Mining of MEDLINE

DIP has recently started to use data mining for finding relevant publications in MEDLINE. A Bayesian classifier extracts abstracts that potentially describe protein-protein interactions, and these articles are then checked by a (human) curator[138]. This approach might rapidly increase the number of interaction entries in DIP and this novel idea is the main reason I have included DIP in this compilation.

5.4.3 Interact – a protein-protein interaction database

Interact is an object-oriented database to accommodate and query data associated with protein-protein interactions[139]. The database is implemented in the commercial OODBMS

[135] http://www.mdsproteomics.com. The information concerning the co-operation with IBM can be found at http://www.mdsproteomics.com/default.asp?qID=8&qType=PressDisplay, "*IBM and MDS Proteomics"* (2001-12-10).

[136] Whether BIND shall continue to be free of charge is not mentioned, and neither is the relation between the publicly available BIND and MDS Proteomics proprietary version of BIND (used in the company's own functional proteomics research projects).

[137] Xenarios, Fernandez, Salwinski, Duan, Thompson, Marcotte and Eisenberg 2001; Xenarios, Rice, Salwinski, Baron, Marcotte and Eisenberg 2000.

[138] Marcotte, Xenarios and Eisenberg 2001.

[139] *Interact - A Protein-Protein Interaction database* (2001-12-10).

Poet 5.0. This has the advantage that the DBMS is fully ODMG compliant, and that it can be queried by OQL. Such OQL queries can be embedded into application programs, or entered through a web form.

According to the web page of Interact there are about 1000 interactions and 200 complexes in the database. These data have either been entered by scientists through web forms or loaded from MIPS[140].

The database Interact is presently *not* publicly available from the web site, not even for a demo - probably because the project has been terminated.

The database schema of Interact is given as a UML class diagram on the Interact web page. A simplified version of the schema is given in Figure 13. This is similar to a class diagram for protein interaction data published by the same group in connection with the GIMS project (see section 6.3.2).

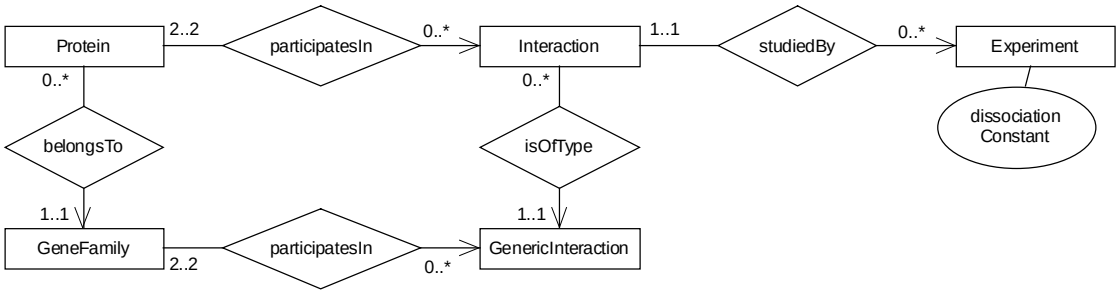


Figure 13. Simplified schema of how protein interactions are represented in Interact.

5.4.4 The Binding Database

The Binding DB[141] is a public database of measured binding affinities for various types of molecules, from the biological to the purely chemical. According to the web-site, the BindingDB aims to facilitate a range of activities, including elucidation of the physical mechanisms of molecular recognition, discovery of new drugs, and discovery of ligands for use in chemical separations and catalysis.

Data is publicly available through web form searches on, e.g., one or two reactant names, authors, and ΔG^{o} ranges. It is also possible to perform "advanced queries" to combine a selection of search fields, or to perform BLAST searches. A very nice feature of the Binding DB is the structure search. The user can sketch a molecular structure on a canvas and search for the structure in the database.

To the best of my knowledge, no information on the architecture or database solution is available on the web site. However, a detailed term catalogue[142] *is* available, with names and explanations for entity types and attributes. From this metadata it seems the web-site is backed by a relational database. A DTD for XML data is also available.

[140] Mewes, Hani, Pfeifer and Frishman 1998.

[141] The Binding DB is managed by the Center for Advanced Research in Biotechnology, University of Maryland, see *The Binding Database*, http://www.bindingdb.org, (2001-12-09).

[142] http://www.bindingdb.org/bind/entity_report.html (2001-12-10).

The database presently holds only few data and the rate of growth the last half-year has been insignificant[143]. A further disadvantage of the database is that the data is not curated. Experimentalists are invited to deposit binding data via on-line forms, with the only quality restriction being that the method by which data was determined must have been published.

The main lesson to learn from the Binding DB is the use of a drawing tool to help in substructure searches.

5.4.5 PDSP Drug Database

The Psychoactive Drug Screening Program (PDSP) keeps a database of dissociation constants (K_i values) for receptor/ligand complexes. It is possible to search on, e.g., receptor, radio-ligand ("hot ligand"), test ligand (competitor), and combinations thereof.

Some (in-house) experimental information is available on-line, and all dissociation constants have links to PubMed literature entries (see section 5.5.1). The database is publicly available through a web form, but there seems to be no mechanism for user submission of data.

The data in the database seems highly relevant to the PAQS project, but there is no description of the long-term goals, the architecture or the database schema on the web site.

5.4.6 GPCRDB

GPCRDB[144] is an informal collaboration between different providers of databases for G protein-coupled receptors (GPCRs). The data can be queried through various web forms, and the focus on G-coupled receptors fit nicely with the focus of present proteo-chemometric research in Uppsala.

Of interest is also that GPCRDB replies to "advanced queries" by using an embedded "smart query engine"[145]. The system supports query rewrite mechanisms which, in case the first query results obtained by the user are not satisfactory, propose more restricted or more relaxed reformulations of the queries. This advanced query system is implemented with an object-relational Informix Universal Server (IUS) DBMS.

5.5 Other Bioinformatics Data Sources on the Web

In the previous section we saw a few publicly available databases on the Web which specialise on interactions between macromolecules and small molecules. There are, however, many important databases for useful background information, such as protein sequences, protein 3D structures, and literature references. Some of these will be described in this section.

An important aspect of information systems in biology is that there are primary and secondary databases. The primary databases are "archives", they store experimental results, possibly with some interpretation. On the other hand, the data of secondary databases are *curated*, they

[143] 239 binding reactions 2001-12-10, and 236 binding reactions 2001-04-10.

[144] *GPCRDB*, http://www.gpcr.org/ (2001-12-10).

[145] *GPCR Query*, http://www.darmstadt.gmd.de/~gpcrdb/ (2001-12-10); Che, Chen and Aberer 1999.

have gone through some independent review process, and are generally considered to be of high quality.

In bioinformatics, the three domains of interest which has been most subjected to database storage are nucleotide sequences of genes, amino acid sequences of proteins, and protein 3D structures (see, e.g. Baxevanis and Oulette 2001).

5.5.1 PubMed and MEDLINE

PubMed is a service of the National Library of Medicine (NLM, http://www.nlm.nih.gov/) to provide the public with access to over 11 million MEDLINE citations and additional life science journals. PubMed includes links to many sites providing full text articles and other related resources. MEDLINE, in turn, is NLM's main bibliographic database, covering the fields of medicine, nursing, dentistry, veterinary medicine, health care systems, and preclinical sciences.

PubMed is publicly available via the NCBI Entrez retrieval system[146]. Obviously, it would be of great advantage if PAQS could interface easily to PubMed. A minimum requirement is that all literature citations should have "MEDLINE unique identifier" (MUID) or "PubMed identifier" (PMID), where appropriate.

5.5.2 DDBJ/EMBL/GenBank

During the last years the sequencing of the human genome has arisen much public interest. Although genome data are not directly relevant to the PAQS project, this section will touch upon databases for nucleotide sequences.

Arguably the most important primary database in bioinformatics is the annotated collection of all publicly available nucleotide and protein sequences held by GenBank (USA), EMBL (European Molecular Biology Laboratory), and DDBJ (DNA DataBank of Japan). These three databases take part in the International Nucleotide Sequence Database Collaboration and exchange information daily. Thus, all three contain the same sequences, but provide them on slightly different formats.

GenBank

GenBank is the genetic sequence database of NIH (the National Institute of Health), built by NCBI (The National Center for Biotechnology Information). Users retrieve data on the GenBank flatfile format (GBFF)[147], but as described in section 5.2.2 efforts are underway to provide data on XML format (mapped from ASN.1).

EMBL at EBI

The European Bioinformatics Institute (EBI, http://www.ebi.ac.uk) is an outstation to the European Molecular Biology Laboratory (EMBL). EBI manages a range of databases of biological data, including nucleic acid sequences, protein sequences, and macromolecular structures.

Although the EMBL nucleotide sequence database is implemented in an Oracle8i database system normal users have traditionally only been able to retrieve data on the "EMBL flatfile

[146] *Entrez PubMed*, http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=PubMed (2001-12-18).

[147] Karsch-Mizrachi and Oulette 2001.

format". The XEMBL project[148] tries to provide users with EMBL nucleotide sequences on various XML formats (see section 5.2.4) formats. Presently, BSML and AGAVE are available.

Thus, we see that EBI and NCBI here follow different strategies. EMBL provide users with data on XML formats of other organisations, while NCBI simply maps their internal ASN.1 to a new (site-specific) XML format.

5.5.3 SWISS-PROT and ExPASy

The ExPASy (Expert Protein Analysis System, http://www.expasy.ch/) server of the Swiss Institute of Bioinformatics (SIB) is mainly concerned with the analysis of protein sequences and structures. ExPASy is a portal to at least eight different databases, where the most important are SWISS-PROT and TrEMBL.

SWISS-PROT[149] is a curated protein sequence database which strives to provide a high level of annotations[150], a minimal level of redundancy (different literature reports are merged as much as possible) and high level of integration with other databases (cross-references to about 60 other databases).

The database was created in 1986, and is maintained by SIB and EBI (European Bioinformatics Institute) in collaboration. Release 40.6 of SWISS-PROT (13-Dec-2001) contains 103258 entries. SWISS-PROT can be searched through web forms, by accession number, author, citation, organism, gene name, et cetera. Users may alternatively retrieve data on SWISS-PROT flat file format by means of ftp.

TrEMBL is a computer-annotated supplement of SWISS-PROT that contains all the translations of EMBL nucleotide sequence entries which have not yet been integrated in SWISS-PROT.

SIB has, together with EBI, formed the Human Proteomics Initiative (HPI) project[151]. The goal of HPI is to high quality annotations of all known human protein sequences.

5.5.4 PDB

The Protein Data Bank (PDB)[152] has long been a repository for protein 3-D structures determined by X-ray crystallography. The data format (also called PDB) is popular in many branches of chemistry, not only crystallography. Nowadays, PDB also contains structures determined by NMR, and recently the database was transferred from Brookhaven National Laboratory to RCSB (the Research Collaboratory for Structural Bioinformatics, http://www.rcsb.org/).

---

[148] *The XEBML Project*, http://www.eb.ac.uk/xembl (2001-12-06).

[149] *ExPASy - SWISS-PROT and TrEMBL*, http://us.expasy.org/sprot/ (2001-12-17).

[150] SWISS-PROT annotations of a protein are descriptions of its function(s), post-translational modification(s), binding sites, 2º and 4º structures, variants, similarities to other proteins, and disease(s) associated with deficiencies in the protein, et cetera.

[151] *Human Proteomics Initiative*, http://www.expasy.org/sprot/hpi/ (2001-12-06).

[152] Berman, Westbrook, Feng, Gilliand, Bhat, Weissig, Shindyalov and Bourne 2000; *The RCSB Protein Data Bank*, http://www.rcsb.org/pdb/ (2001-12-17).

RCSB has changed the PDB data format from the classical "Fortran-style" flat file format to mmCIF (section 5.2.5), which, however, is transparent to users accessing the traditional format. Since different research groups use (or interpret) the PDB format differently it might be advantageous for a new project such as PAQS to try to use the stricter mmCIF format.

Primary experimental and coordinate data are stored under a relational Sybase DBMS. However, users may only access data through web forms, or retrieve the final curated data by ftp (on PDB or mmCIF format).

As of 11-Dec-2001 PDB contained 16859 structures. This is about one order of magnitude less than the number of entries in SWISS-PROT, and this is due to the fact that it is much more difficult to experimentally determine 3-D structures than amino acid sequences.

5.5.5 MMDB

The Molecular Modelling DataBase (MMDB) contains protein 3-D structures derived from PDB data[153]. However, MMDB differs from PDB in several aspects. First of all, in PDB structures are described as atomic coordinates, which can be supplemented by information on which amino acid residue each atom belongs to. However, it is up to the application program to determine which atoms that are bonded to each other. In MMDB the data files contain all bonding information explicitly[154].

A second difference is that MMDB (as other NCBI databases) uses ASN.1 as a schema definition language. Thus MMDB is well integrated with the US National Library of Medicine and GenBank. MMDB is accessible through the Entrez system (section 6.4.1).

5.5.6 PIR

The Protein Information Resource (PIR, http://pir.georgetown.edu) distributes PIR-International Protein Sequence Database (PSD) together with MIPS of Germany and JIPID of Japan. PIR-PSD[155] claims to be "the most comprehensive and expertly annotated protein sequence database in the public domain".

PIR-PSD is a public domain database, accessible through web forms and downloadable by ftp. The PIR-PSD flat files are available in XML format, with an associated DTD file, as well as in the original NBRF and CODATA formats. The PSD sequence file is distributed in FASTA format.

---

[153] Hogue 2001.

[154] From a purist's point of view the PDB approach is the correct. It is the atomic coordinates which are determined in X-ray crystallography, and there are furthermore a few cases where it is not straightforward to determine if two atoms are bonded together or not.

However, in software for modelling and visualisation of biomolecules it is often necessary to define exactly which atoms that are bonded together (and by which types of bonds), and it is probably from this practise the need for MMDB has arisen.

A further advantage of the MMDB approach is that a database search for a specific (simple) structural feature might be faster in internal coordinates than in Cartesian coordinates. This, however, will depend on the structural feature.

[155] *PIR-International Protein Sequence Database*, http://pir.georgetown.edu/pirwww/dbinfo/pirpsd.html (2001-12-16).

PIR-PSD and other databases at PIR are implemented under an object-relational Oracle8i DBMS. Barker et al (2001) describe that they have used both object and relational features in the database design, and also both ER and UML modelling. Unfortunately, there seems to be no detailed information (e.g. database schemas) available in print or on the PIR web site[156].

5.5.7 GDB - the Genome DataBase

The Genome DataBase (GDB)[157] is the official central repository for genomic mapping data resulting from the Human Genome Project. Although the contents of the database is not of direct interest to the PAQS project some other aspects of GDB are.

GDB is implemented in the OPM data model (vide infra). The web site of GDB has a lot of relevant information, e.g. database schemas and descriptions of the data model and the system architecture. This is in very good accordance with the conclusions drawn by Markowitz et al (2001) in section 5.1[158].

The data stored in GDB are divided among three databases, one for biological information, one for citations, and one "registry" database for people with editing privileges or otherwise involved in the database. This modular approach is managed by an object broker middle layer, between clients and the Sybase data servers.

Thus, I think PAQS should take over the idea of a separate server with people and authorisations from GDB. This should be possible to implement as a separate Amos II server, although Amos II presently does not support protection, neither of objects nor of the type system. Furthermore, GDB has set an example when it comes to meta-data availability. (However, this meta-data is only available as documents. The actual database is not queried.)

OPM - the Object-Protocol Model

The Object-Protocol Model[159] was developed by Markowitz, Chen et al[160] as an object-oriented data model with specific constructs for representing scientific experiments. Experiments are modelled by protocol classes (instead of object classes). A protocol takes an input and produces an output. Further, protocol classes can be expanded as alternative subprotocols, sequences of subprotocols, and optional subprotocols.

There are a range of database management and browsing tools available for OPM. However, the OPM project finished in 1997, and the members went into industry. This is a potential problem for GDB and other projects where a unique DBMS is used to implement a database system which is meant to last for a long time. If the OPM project partners are no longer available, and if the GDB staff has not got enough expertise in the OPM system some modifications of the database system will be difficult to perform. E.g., changes of the database schema and of input and output formats are tasks the GDB staff should be able to do, but they probably cannot improve or extend the system (e.g. implement new features of SQL) without technical OPM expertise.

[156] Furthermore, although Barker et al (2001) claim that metadata and technical bulletins are available at the PIR web site, I have not been able to find any such material produced more recently than 1997.

[157] *The Genome Database*, http://www.gdb.org (2001-04-20).

[158] The OPM model was developed by the very same research group, and this group has also taken part in the construction of GDB.

[159] Chen and Markowitz 1995.

[160] *The OPM Project*, http://gizmo.lbl.gov/opm.html (2001-12-17).

6 Related Work: Database Systems for Life Science Data Integration

In this chapter a few database systems for integration of biological, bioinformatics, or life science data will be described. The purpose is to compare the approaches and capabilities of these systems with those anticipated for PAQS, and perhaps to induce ideas about new features for PAQS.

Some major efforts for the integration of biology data were mentioned already in chapter 5, treated as potential data formats for PAQS, e.g. the two CORBA standardisation project mentioned in section 5.3.

There is a rapid growth in the number of niched companies performing proteomics or drug design research, "out-sourced" from a (larger) pharmaceutical company. Such "proteomics companies" often claim to have unique expertise in one or several fields (e.g. mass spectrometry, cell signalling, protein structure determination, or computational chemistry). In a way, these companies function as mediators: they integrate information from different sources, and add value in the form of their own research. However, since we are interested in software platforms which facilitate information integration we will not further discuss such companies.

As discussed in the beginning of chapter 5, only some of the many integration projects can be described in this Thesis.

6.1 Research Prototype Mediator Systems

Most mediator systems for the integration of biological data have the characteristic in common that they origin in mediator research projects performed at an academic computer science laboratory, and that the interest in using these mediators for biological data has grown during the bioinformatics boom of the 1990's. Of special importance is the well-known Human Genome Project (http://www.ornl.gov/hgmis/), which is reflected by the fact that most of the projects aim to integrate different sources of genome data.

In principle, this type of mediator systems could be used as data sources in the PAQS project, or conversely. Thus, we *could* have a system of collaborating mediators, similar to the picture given in Figure 10, but with the difference that all participants would not be Amos II mediators. From a technical point of view it is the public interface which is determining. If a mediator system only publishes web forms for fixed-form queries other systems will only have limited use for it. If, on the other hand, a mediator is open for standard OQL queries, has a CORBA interface, or can be queried as an XML source, there is greater potential for collaboration.

6.1.1 P/FDM

P/FDM (Prolog/Functional Data Model) is an OODBMS created by Gray, Kemp et al in Aberdeen[161]. Similarly to Amos II, P/FDM employs a functional data model based on DAPLEX (see section 3.2.5).

[161] Gray and Kemp 1990.

The Aberdeen group has used P/FDM for biological data for more than 10 years. Their main work seems to have been on defining schemas for representing protein structures and the integration of such data from heterogeneous data sources[162]. For example, a declarative constraint language to describe the semantics of 3-D protein structure data has been developed[163]. An important and interesting goal of the Aberdeen group has been to make it possible to perform efficient geometric calculations over proteins *in* the P/FDM system[162].

As mentioned above, P/FDM has been used as a CORBA server[164]. An architecture is described where a graphical user interface connects to P/FDM through a coarse grain CORBA interface, and P/FDM provides query processing services over local and remote databases, also through CORBA interfaces. It seems that only the "upper", GUI-to-P/FDM, part of the architecture was implemented, perhaps due to the lack of CORBA servers at the time.

The most important data source for P/FDM seems to have been SRS (Sequence Retrieval System, see section 6.4.2), but recently the group has widened the scope to more diverse data sources[165], e.g. EBI's ArrayExpress (microarray based gene expression data, see section 6.3.1) and Ensembl (eukaryotic genomes). A recent publication[166] on P/FDM describes how the system is used to build a database federation for bioinformatics, and how the P/FDM mediator is used to integrate locally stored data with data from the SRS.

To conclude, the P/FDM effort is similar to the PAQS project when it comes to architecture and technical possibilities. However, the applications lie within another domain, and the data sources are only partially overlapping.

6.1.2 K2/Kleisli

K2 (and its predecessor Kleisli) follows a mediating, view integrating, approach to the integration of heterogeneous data sources[167]. In the K2 architecture wrappers are called "data drivers", and such have been developed for a range of formats and sources, e.g., GenBank, BLAST, KEGG, and SRS. These data drivers have the responsibility to provide K2 with source metadata, to transmit queries to the sources, and to convert the query results to K2's internal format. However, rewriting from OQL (the query language of K2) to the native query language of the source is made by K2, not by the data driver.

K2 has an extensible rule-based optimiser, but optimisation is presently not cost-based due to problems in estimating accurate costs in the distributed environment. An interesting option in K2 is to define virtual classes which span several underlying databases. This resembles the integration union types of Amos II[168]. A major advantage of K2 over the earlier Kleisli, and possibly over Amos II, is that K2 has OMG's standard OQL as query language.

From the examples given by Davidson et al (2001) it seems the scope is wider than only gene and protein sequences. The overlap with the PAQS project seems presently small, however.

[162] Kemp, Dupont and Gray 1996.

[163] Embury and Gray 1995.

[164] Kemp, Robertson, Gray and Angelopoulos 2000.

[165] Kemp, Fothergill, Gray and Angelopoulos 2001.

[166] Kemp, Angelopoulos and Gray 2000.

[167] Davidson, Crabtree, Brunk, Schug, Tannen, Overton and Stoeckert 2001.

[168] Josifovski and Risch 1999.

6.1.3 Tambis and Ontology-Based Approaches

In computer science the term *ontology* is often used for a "specification of a representational vocabulary for a shared domain of discourse, which may include definitions of classes, relations, functions, and other objects" (Kashyap and Sheth 1999). Thus, an ontology can be seen as a list of commonly understood terms, and different application domains have different ontologies[169].

An ontology in this meaning of "concept repository" can be used for information integration, and one representative example of this is the TAMBIS project described in the following subsection. Another example is mmCIF (section 5.2.5), which has been described as an ontology for macromolecular structure.

TAMBIS

The TAMBIS project from Manchester[170] aims to provide users with maximum transparency when accessing bioinformatics data sources around the world. To achieve this TAMBIS uses a central ontology (a conceptual representation of biological concepts and terminology) together with mappings from terms of the ontology onto terms in external sources. TAMBIS follows a mediator/wrapper architecture where query translation, query planning, and wrapping is based on CPL/Kleisli[171].

The central and most developed part of the TAMBIS project seems to be TaO, the Tambis Ontology. TaO has several roles: (i) to describe biologists knowledge, (ii) to encompass the schemas of underlying data sources, (iii) to link conceptual terms to terms in the sources, (iv) to mediate between equivalent or nearly equivalent concepts in different sources, and (v) to help users to form biologically realistic queries and explore the ontology.

TAMBIS focuses on DNA and protein structures. Representative questions a user can ask is "find homologues to apoptosis receptor proteins" and "find motifs in enzymes which use thyamine as substrate and iron as cofactor". Higher-level, exploring, questions like "what can I say about the concept 'receptor'?" can be answered, too.

A Difference Between the Ontology Approach and Mediation

TAMBIS has a mediator/wrapper architecture similar to that of e.g. Kleisli, P/FDM, or PAQS. However, when it comes to information integration there is a major difference between the mediation approach and the ontology approach. In PAQS we will only build a partial integrated schema of the sources. I.e., we will only care to model those aspects of the domain that is of interest to the users of PAQS. In an ontology-based approach, the ontology should (eventually) encompass all concepts that the user community knows about the domain (for example bioinformatics). I.e., the ontology could be said to correspond to a global schema, and *all* information present in the source schemas should also be present in the ontology[172].

[169] In philosophy, on the other hand, ontology is a form of metaphysics dealing with the true nature of being (Odelstad 2001).

[170] *TAMBIS*, http://img.cs.man.ac.uk/tambis/index.html (2001-12-06); Baker, Goble, Bechhofer, Paton, Stevens and Brass 1999.

[171] Buneman, Davidson, Hart, Overton and Wong 1995.

[172] In a way, we could say that the ontology represents a "super-global" schema, since there will probably be concepts in the ontology which are present in none of the sources. However, a mediator may contain local data not present in any source, and often this local data has some semantic meaning which no source schema captures. Thus, the same "super-globality" applies for mediators.

6.2 Commercial Middleware Solutions

In principle we can think of two types of commercial systems. The software can be intended to be installed at the customer's site, and the business model would be to sell (or rent) a system, and to sell consulting expertise. This applies for DiscoveryLink, described in the next section.

A second business model would be to integrate data at the software provider's site, and publish a web interface which customers can put queries through[173]. This has the serious disadvantage that proprietary data owned by the customer is not included in the integration.

It may be noted that the bioinformatics research community has a strong tradition of sharing data, see, e.g., the publicly available gene and protein sequences in the data sources of section 5.5, and it would be difficult to sell web services to academic users. This contrasts with the situation in, e.g., chemistry.

6.2.1 DiscoveryLink

DiscoveryLink[174] is a database middleware system from IBM Life Sciences[175]. The purpose of DiscoveryLink is very similar to that of the PAQS project[176] and the research projects described in section 6.1. The system integrates heterogeneous data sources so that users can access data in a uniform and transparent manner.

Like the research prototypes of the previous section DiscoveryLink follows a mediator/wrapper-like architecture (see Figure 14). However, DiscoveryLink uses IBM's DB2 UDB as mediator (or middleware) and employs a relational data model for information integration. The research prototypes (with object-oriented or functional data models) ought to provide richer modelling capabilities in the mediators. However, IBM argues that the use of a commercial industrial-strength DBMS as middleware is of advantage. Furthermore, users can query the DiscoveryLink system with standard SQL queries while most similar systems use OQL or some specialised query language.

Several enhancements to the way DiscoveryLink now works are described by Haas et al (2001). One of these is the possibility to keep prematerialized summary tables stored in DiscoveryLink to avoid accessing the data sources. This is probably an idea which should be taken over by the PAQS project.

[173] A number of alternatives are possible here. Pay-per-minute, pay-per-month, pay-per-query, and pay-per-tuple-returned are probably all too simple. For example, FIZ Karlsruhe (http://www.fiz-karlsruhe.de/) uses a combination of time connected, number of queries, amount of information returned, and a cost factor specific for the underlying source to bill customers.

[174] *DiscoveryLink* (2001-12-18); Haas, Schwarz, Kodali, Kotlar, Rice, and Swope 2001.

[175] *IBM Life Sciences* (2001-12-13).

[176] Actually, one of the example queries used by IBM to demonstrate how DiscoveryLink works is "Show me all the compounds that have been tested against members of the family of serotonin receptors and have IC50 values within nanomolar/ml range". Apart from the unfortunate choice of unit for IC50 this is exactly the kind of query we would like to put to PAQS.

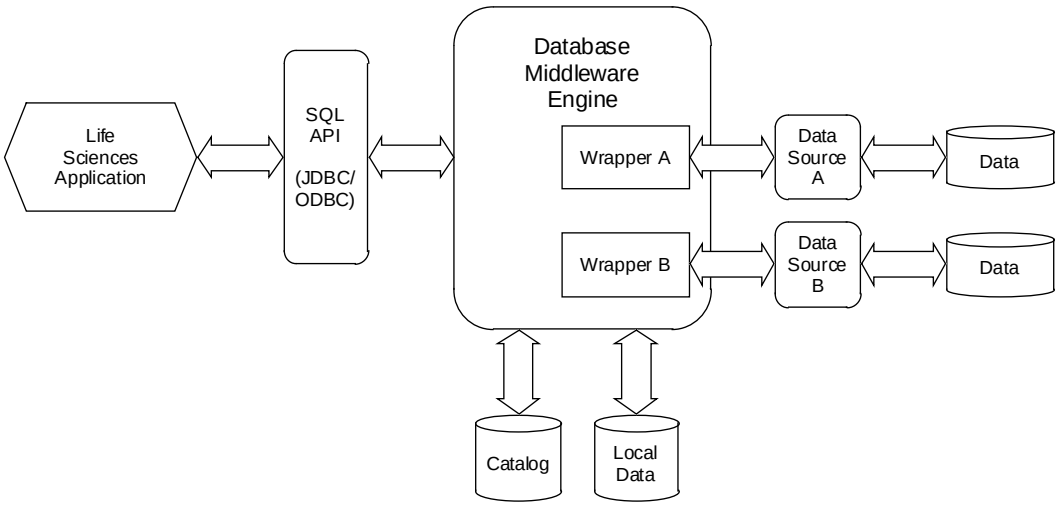


Figure 14. DiscoveryLink architecture. (After Haas et al 2001.)

Wrappers

The data sources are integrated with the system by means of wrappers (see section 4.4). This solution is taken from IBM's research project Garlic[177]. A wrapper has four principal responsibilities: (i) To map the schema of the data source to DiscoveryLink's relational model, (ii) to inform DiscoveryLink of the data source's query capabilities, (iii) to map query fragments from DiscoveryLink to requests in the native query language of the data source, and (iv) to issue those query requests and return the results to DiscoveryLink. An important feature is that any specialised query capabilities of a data source are modelled as user-defined functions in DiscoveryLink.

A future enhancement to DiscoveryLink is to improve query optimisation by directly involving the wrappers in query planning. When a user issues a query to DiscoveryLink the system should ask the wrappers which parts of the query each of them is able to answer. In return, the wrappers should send a "wrapper plan" together with the associated estimated cost. The DBMS then uses the wrapper plans to construct a reasonable global query plan. This solution to query optimisation over distributed heterogeneous data sources has been validated in the Garlic project, and is more or less identical to the approach in the new SQL/MED standard (see section 4.4.3).

Semantic Data Integration

Although DiscoveryLink can integrate data from different sources the system does not automatically solve the problems of heterogeneity[178]. A solution suggested by Haas et al (2001) is that database administration staff should build translation tables, which could then be stored in DiscoveryLink. The lookup in such tables would impair performance, but the solution is in principle possible for all middleware or mediator systems which have the capability to store data locally.

[177] *Garlic*, http://www.almaden.ibm.com/cs/garlic.html (2001-12-18).

[178] Haas et al (2001) give three examples of "semantic conflicts": (i) upper/lower case strings, (ii) different names for one and the same drug, and (iii) no common keys between objects in different sources. In the terminology of section 4.3.3 (i) is a data conflict, while (ii) and (iii) are identity conflicts. A semantic conflict, on the other hand, would be when a concept is interpreted differently in different sources. (E.g., if one source treats ligands and radiolabelled ligands uniformly as "ligands", while another separates them into "ligands" and "hot ligands".)

Schema Evolution

Haas et al (2001) discuss the frequent change of database schemas in life science data sources. They describe how the wrapper architecture of DiscoveryLink has been designed for extensibility, and that the system therefore is well prepared for changes in data sources. In this context it is worth noting that the easier schema evolution is said to be one of the advantages of object-oriented DBMSs over relational DBMSs[179]. This might lead one to believe that mediators based on OO (or functional) data models are better suited for a situation where source schemas changes are frequent. However, it is the wrappers which should do the mapping between source and middleware, and it is of course essential that the wrappers are easy to modify. In practise, it should be the case that small changes in data source schemas can be absorbed by the wrappers, while only larger source changes require changes to the middleware schema.

Availability

DiscoveryLink is a commercial product of IBM. From the demonstration examples available at the DiscoveryLink web site I conclude that it should be possible to query the data sources through some web interface once the system has been installed at the customer's site. Another, and potentially more useful way to query DiscoveryLink is through DB2's JDBC API. There is no mentioning of constructing a publicly available web site powered by DiscoveryLink.

DiscoveryLink is a major component of the I3C architecture, see section 5.3.4.

6.2.2 Oracle Gateway with DARWIN Analysis

Banerjee (2000) describes how Oracle's data mining tool Darwin can be used for bioinformatics and proposes Oracle's gateway technologies as a means for accessing non-Oracle databases[180].

In an independent (but perhaps slightly dated) study of three (anonymous) commercial product Rezende and Hergula (1998) found that the gateway approach to integrating heterogeneous data sources was inferior to more general middleware approaches.

In my opinion, Banerjee's suggestion of having Oracle technology as a powerful back-end for web servers is more realistic. Such a portal could then, for example, provide wide research communities with access to XML based data.

6.3 Warehouse Approaches

From chapter 4 we recall that a data warehouse is a central repository for integrated data. Several of the data sources listed in sections 5.3.4 and 5.5 include components of integration by the warehousing approach and could have been discussed in this section instead.

[179] Connolly and Begg 2002, p 792, 828.

[180] To the best of my knowledge, this system - adapted for bioinformatics - is not commercially available. However, since the solution was presented by an Oracle affiliate and consists of commercially available components, I have chosen to list it in this section.

6.3.1 ArrayExpress

At EBI (European Bioinformatics Institute) the ArrayExpress project is committed to establishing a public repository for gene expression data from microarray experiments[181]. Microarray data are not of direct use to the PAQS project[182] but the ArrayExpress project is nevertheless worth studying. First of all, ArrayExpress attempts to define a standard for a subdomain of experimental molecular biology. Similarly, one possible goal of PAQS is to define a standard for binding assay data. On a lower level, there may be similarities in how experiments are described, e.g. how an array or an assay is represented in a database.

ArrayExpress aims to build a data repository. The purposes of the repository are[183] to make the data available to many parties, to facilitate "cross-validation" of data, to establish benchmarks and standards, and to create the ability to build up progressively more detailed information. Finally, the repository should function as a public resource which can be referenced by scientific literature[184].

Obviously, a repository of data follows the warehousing approach to information integration, described in section 4.1. An interesting aspect of this is of course that ArrayExpress will need to define a database schema which is expressive and detailed enough to capture information from a wide range of laboratories and experimental set-ups. This would be a great challenge even for a "traditional" information domain, but an additional complication is that the microarray scene is new, and rapidly changing. Thus, the schema as well as the data model used for constructing the schema and for implementing the database must be flexible enough to provide for facile schema evolution.

A conceptual schema in UML is given on the home page of ArrayExpress[185]. Interestingly, this schema lacks all object-oriented features, and it seems to be intended for implementation in a relational DBMS[186]. The schema has been implemented in P/FDM (see section 6.1.1)[187], and as a relational database[188].

The ArrayExpress homepage also links to a more comprehensive description of how microarray data can be represented and stored[189]. This document has schemas drawn in

[181] *The ArrayExpress Database* (2001-12-17).

[182] First of all, the purpose of microarrays is to investigate what proteins are produced from the different genes of a genome under different environmental conditions. In PAQS, we are interested in how proteins interact with ligands. Secondly, the experimental set-up is different and the experimental raw data are different too (fluorescent or radiation images in microarrays, versus radioactive counts from a detector in binding experiments). Finally, the analysis of the experiments are different.

[183] *Establishing a Public Repository for DNA Microarray-Based Gene Expression Data* (2001-12-17).

[184] Repositories with this reference function exist for example in crystallography. In that scientific community it is customary to publish (in a journal paper) only a drawing of a substance whose structure has been determined, together with some important geometric parameters (e.g. bond lengths). A full list of atomic coordinates and thermal parameters must then be deposited, for example with the Cambridge Crystallographic Data Centre (CCDC, http://www.ccdc.cam.ac.uk (2001-12-10)). Other depositing schemes do exist, too.

[185] *Structure and Design of ArrayExpress Database* (2001-12-17)

[186] In the document is stated that it is straight-forward to map the schema to the ER model, but also to a range of other models (e.g. Java programs).

[187] *aeFDM: FDM implementation of the ArrayExpress Schema* (2001-12-17).

[188] *The maxdSQL Database* (2001-12-17).

[189] *ArrayExpress Schema, Model and Documentation* (2001-12-17).

OMT[190], and *does* include OO features (e.g. subtyping). The schemas of references 185 and 189 differ in central parts.

6.3.2 GIMS - the Genome Information Management System

In a recent paper[191] which mainly deals with how genomic data, protein interactions, and gene expression experiments can be modelled conceptually Paton et al introduce the Genome Information Management System (GIMS)[192]. From the brief description it seems GIMS is a continuation of Interact (section 5.4.3), with a much wider scope.

GIMS follows a warehousing approach to integration, with the warehouse implemented in the OODBMS Poet. Data from "information sources" are loaded through wrappers. The principal way to explore GIMS is expected to be through canned queries, i.e. predefined parameterised queries in web browsers or application programs.

One of the conceptual models of Paton et al (2000) deals with how transcriptome data can be represented (see Figure 15). This is a similar modelling problem as that of ArrayExpress (section 6.3.1), and the schema of Paton et al has given some inspiration to the modelling of experiments described later in this Thesis.

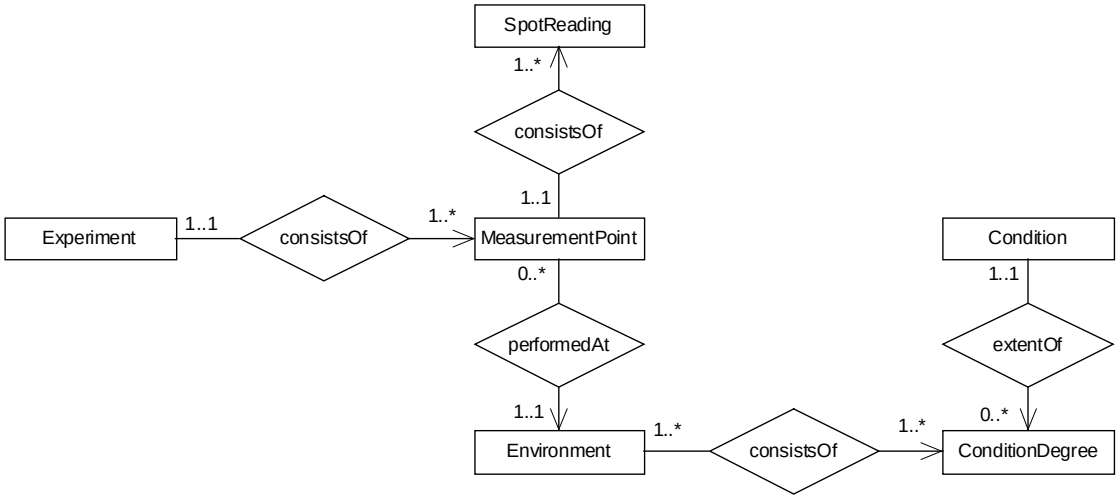


Figure 15. Schema for representing transcriptome data according to Paton et al (2000), simplified and transferred to the functional diagram notation.

Each experiment has a collection of measurement points, consisting of spot readings (in the special case of gene expression data). To each measurement point we may associate an environment, which records the extent (condition degree) to which some property (condition) holds when a measurement is made[193].

[190] OMT (Object Modeling Technique) is a diagrammatic notation for object-oriented modelling. OMT is similar to (and a predecessor of) UML.

[191] Paton, Khan, Hayes, Moussouni, Brass, Eilbeck, Goble, Hubbard.and Oliver 2000.

[192] *Genome Information Management System*, http://img.cs.man.ac.uk/gims/ (2001-12-17).

[193] For example, the two conditions temperature and pH may have the values 25 ºC and 7.0, respectively. These two "condition degrees" then constitute the "environment" valid for a specific set of measurement points.

6.3.3 GUS - the Genomics Unified Schema

Davidson et al (2001) have developed the Genomics Unified Schema (GUS), which is a data warehouse used to integrate data from major sequence databases (GenBank, SWISS-PROT, dbEST).

GUS uses a relational data model and is implemented with an Oracle8i DBMS. The relational schema is large, over 180 tables. One reason for this is that it is difficult to represent the compressed schemas of sequence databases by a relational model. Only SWISS-PROT maps to 15 tables, and all in all about 50 tables are required to mirror the external databases. The large schema makes the system difficult to comprehend, and therefore an object layer (in Perl) has been implemented on top of the relational system.

In conventional data warehouses for business applications value is added to the data through summations on different levels of aggregation. In GUS, which has a focus on sequences and genes, it is annotations that add value. An important aspect of GUS is hence to track how annotations change, when they change, which algorithms that were used, and so on. This management of metadata is handled by version tables.

6.4 Interlinked Collections of Data Sources

One approach to providing a useful query interface to a collection of data sources is to create a layer of *links* between database records on top of the data sources. Thus, if a user searches for the protein X, she will not only get the sequence from a protein sequence database, but also the corresponding records from a literature database, and a 3-D structure database - if she has marked those databases "active" in the web interface.

There is no real information *integration* or transparency in this kind of system. The user needs to choose which databases to search, and usually no attempt is made to resolve conflicts. Thus, although the data may be collected at a central site, they reside in separate databases, and are not integrated as in a Data Warehouse. Davidson et al (2001) call this type of system a "*link driven federation*".

6.4.1 Entrez

Entrez[194] is a retrieval system for searching several linked databases. It provides access to biomedical literature (PubMed), nucleotide sequences (GenBank), protein sequences (SWISS-PROT, PIR, PDB and other), 3-D macromolecular structures (MMDB), complete genome assemblies, and several other types of data.

What makes Entrez more powerful than an ordinary web interface is that most of its records are linked to other records, both within a given database and between databases. Records connected through intra-database links are called "neighbours". An important feature is that these links not only are based on exact matches, but also on pre-computed similarity searches.

[194] *Entrez Home,* http://www.ncbi.nlm.nih.gov/Entrez/ (2001-12-18).

6.4.2 SRS

The Sequence Retrieval System (SRS) was developed by EMBL (European Molecular Biology Laboratory) during the 1990s. SRS has gradually evolved and changed, and is now an integration platform for molecular biology and genome analysis, owned by Lion Bioscience[195].

SRS is publicly available at EBI[196] and at several mirror sites. However, some of the accompanying software is not free. There are currently several dozen servers world-wide that provide access to over 300 different databanks via the Web. Only through the EBI web interface a user may access 158 databanks (19-Nov-2001). An SRS server may access databases stored locally together with databases on the Web. Presently, SRS can retrieve data from relational DBMSs, XML servers, and flat files.

SRS integrates heterogeneous data sources ("databanks") behind a single interface and integration framework. SRS is said to employ a "meta level approach" information integration[197]. Thus, SRS uses metadata about structure, format and syntax of the data sources to build indices for each file in each database source, to build link indices between each integrated database, and to help in data retrieval of specific fields upon user requests.

In addition, another level of metadata is used to let users define how data should be retrieved, e.g. as XML data files or as Java objects. Thus, SRS is, according to the information available on the Web, extremely flexible for users who are prepared to invest some time and effort to learn the system and the tools. Furthermore, there are various extensions to SRS, for example, SRS Objects[198] provides APIs to C++, Java, Perl, and Python, as well as CORBA interfaces. It is possible for an organisation to install the SRS system locally (an Oracle 8 DBMS is required) and then to integrate their own data sources.

Integrating Relational Data Sources in SRS

When a new relational database is introduced in an SRS system the administrator needs to run a series of programs to extract the database schema, and manually define a HUB table (vide infra), a HUB accession column (with unique values), and a set of columns that should be presented as search fields in web interfaces.

The querying and retrieval of data from relational databases proceeds in two steps[199]. When a user has filled in the web form to produce a query the first step is to retrieve the relevant keys from a HUB table and the second step is to use the keys to assemble objects with all the fields the user has asked for.

SRS first analyses the schema information it has about the RDBMS and then generates an SQL query which is sent to the RDBMS via JDBC[200]. The interesting part is that the query is formulated so that it converges (through joins) to a "HUB", which is the "table of most interest", and the centre of a star topology[201].

---

[195] http://www.lionbioscience.com/, http://www.lionbio.co.uk/. Actually, the system has changed so much that SRS is now its name, not an acronym.

[196] *SRS 6*, http://srs6.ebi.ac.uk/ (2001-12-17).

[197] *SRS Technology*, http://www.lionbioscience.com/htm/c_1/content_c_1_1_1.htm (2001-04-06).

[198] *SRS 6 Extensions*, http://www.lionbioscience.com/htm/c_1/content_c_1_1_2.htm (2001-04-06).

[199] *SRS Relational* (2001)

[200] JDBC is often interpreted as Java DataBase Connectivity, although this is not an "official" acronym.

[201] Cf. star schemas in Data Warehouse dimensionality design, see, e.g., Connolly and Begg 2002, ch 31.

When the relevant keys (which are drawn from the accession column in the HUB table) have been returned to SRS a new SQL query is formulated to retrieve the information. This query is constructed to start with one or more HUB keys, and then proceed outwards. The relevant information is collected and result objects are built up incrementally.

Although this is not mentioned in the documentation available to me (e.g., refs. 195-199), it is probable that many query plans are stored (or cached) in the SRS system. It seems uneconomical to need to analyse the schemas of the participating databases for each and every standard query.

7 A Database System for Proteo-Chemometric Research

In this section PAQS - the Proteo-chemometric Analysis and Query System - is described. The PAQS project emanates from a research proposal made by Tore Risch (Dept of Information Science, Uppsala University) and Jarl Wikberg (Dept of Pharmaceutical Pharmacology, Uppsala University), and should result in a database system for proteo-chemometric research. The approach used in the project is to integrate information from various distributed, heterogeneous, and autonomous data sources by means of a mediator-wrapper architecture.

7.1 Architecture of PAQS

The architecture of PAQS follows that of many other mediator-wrapper systems (see section 4.4). This architecture was defined before the start of my Thesis work. Figure 16 illustrates the PAQS architecture with three layers:

- The *client* layer contains various applications, e.g. for analysis and visualisation. This layer could also contain a web interface.
- The *mediator* layer is the PAQS system itself, the core of which is an extensible database engine, i.e. the Amos II DBMS[202]. The parts which are specific for PAQS are the various wrappers and plug-ins (vide infra).
- The *data source* layer stores the data. Most of these data sources will be resources accessed over the Web, but some will be local databases.

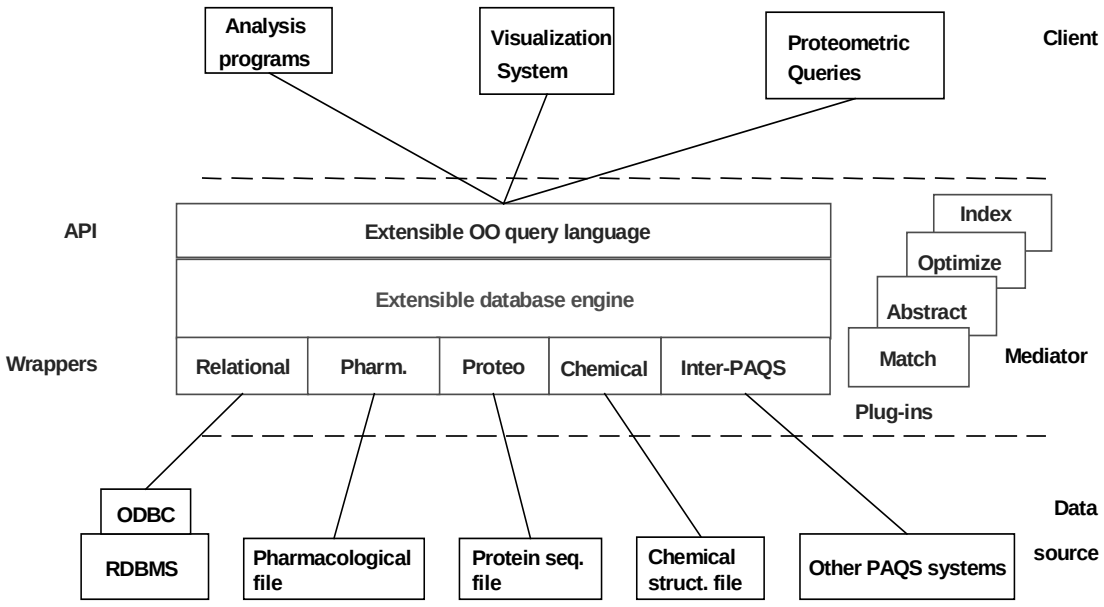


Figure 16. Architecture of PAQS. (Figure made by T. Risch.)

[202] Risch, Josifovski and Katchatounov 2000; Flodin, Josifovski, Katchaounov, Risch, Sköld and Werner 2000.

7.1.1 Application Programs

There are several conceivable types of application programs:

- Analysis programs may access PAQS in order to get hold of needed data. Other analysis programs may take their data from other sources (e.g. from some laboratory equipment) and deposit the results in PAQS.
- Analysis programs could invoke other programs, e.g. for visualisation.
- The system could be open for queries over a web interface, either through some web form with limited querying alternatives or via direct submission of AMOSQL queries.
- Another PAQS system may be a client[203].
- A meta-data browser could be used by persons wishing to understand the PAQS schema.

No work has been done on the client layer for this M.Sc. Thesis.

7.1.2 Wrappers and Data Sources

Wrappers are an important part of the mediator-wrapper architecture of Amos II and PAQS. A wrapper functions as an interface to an external data source, and it is only by means of such wrappers the mediator is able to access external data. The approach has recently been included in the SQL:1999 standard as SQL/MED (Melton et al 2001, see section 4.4.3).

Presently, the Amos II system incorporates wrappers for relational databases (as ODBC data sources) and XML files. It is possible that other wrappers will have to be implemented for access to important data sources. However, this might very well be unnecessary work since there is a very strong trend towards XML formats in life science research. A discussion of various available data sources on the Web was made in chapter 5 of this Thesis. Most binding affinity data sources present their data through web forms, and it is thus important that Amos II can access such forms efficiently[204].

I have not designed or implemented a wrapper in this work. For an example of the use of an existing wrapper, see Appendix G.

7.1.3 Algorithms for Plugging-In

One thing which distinguishes commercial object-relational DBMSs from the older relational DBMSs is the possibility to extend the functionality for a particular information domain by means of "plug-ins" (see section 3.2.4). These possibilities should be even larger for most research prototype ORDBMSs.

In Amos II the foreign functions provide this extensibility. Thus, functionality can be included by code in Java, C, or Lisp. For the PAQS project this is of great importance as many small and middle-sized computations can be performed by foreign functions:

[203] This is conceivable in at least two situations: (i) Several laboratories have their own PAQS systems running, but share local information by accessing each others' systems. (ii) There is one central PAQS installation, but one laboratory wishes to use a local PAQS server to manage some proprietary or preliminary data without sharing it with others.

[204] Petrini 2001.

- A main benefit will be from the point of reusability. I.e., an existing implementation of an algorithm could easily be plugged in and used. There is no need to code and debug the algorithm once more.
- Some algorithms are impossible or very difficult to implement in AMOSQL since the necessary mathematical functions are missing in query languages. However, they can easily be implemented as foreign functions[205].
- Small visualisation programs could be used as plug-ins, e.g. used to do simple xy-plots of data series (section 8.2).
- Foreign functions could work as interfaces to the file system. E.g., we may import a certain data item from a certain type of text file by means of a Java function. (I.e., we construct a very simple kind of wrapper, specialised on a single task.)
- It would even in some cases be possible to wrap a large and complex program by a Java class, and then plug it in by means of a foreign function. One limiting factor will be the response time[206]. Furthermore, if the program requires some additional user input, e.g. through a GUI it is questionable if we conceptually can classify it as a foreign *function*.

An open question in the PAQS architecture is how much of the proteo-chemometric analysis that is to be implemented as AMOSQL functions. In a traditional solution the database system would only be used for data storage and access, through the API. All analysis would be performed in special-purpose programs. At the other extreme is a situation where all analysis and visualisation algorithms are implemented as foreign functions, and the whole proteo-chemometric analysis takes place by calls from the DBMS. The database schemas presented in chapter 8 allows for both these solutions, and intermediaries. In connection with experiment evaluations (section 8.8) I have shown how analysis from PAQS could be implemented.

7.2 The Information Domain with Subdomains

The information domain of interest for proteo-chemometric analysis is coarsely described in Figure 17. The lines indicate direct interdependencies, and one example of how to read the Figure is to follow the bold lines: A series of binding experiments are made with a binding assay. These experiment are then evaluated (by some curve fitting program) so that a number of fit parameters are determined. Each fit parameter represents some property of a chemical entity or a combination of chemical entities (e.g. a binding affinity between a ligand and a receptor).

In chapter 8 of the Thesis the modelling and implementation of most of the subdomains are described. However, all subdomains have not been modelled to equal detail. Subdomains with dotted "borders" in Figure 17 were only implemented as "stubs" (ChemicalEntity, Protocol, Reference), or not at all (Descriptor). These remaining subdomains are further commented upon in section 9.1.

[205] For example, see Appendix C or the matrix operations of Flodin, Orsborn and Risch (1998).

[206] For example, it is not unusual that quantum chemistry computations of molecular properties execute for several days.

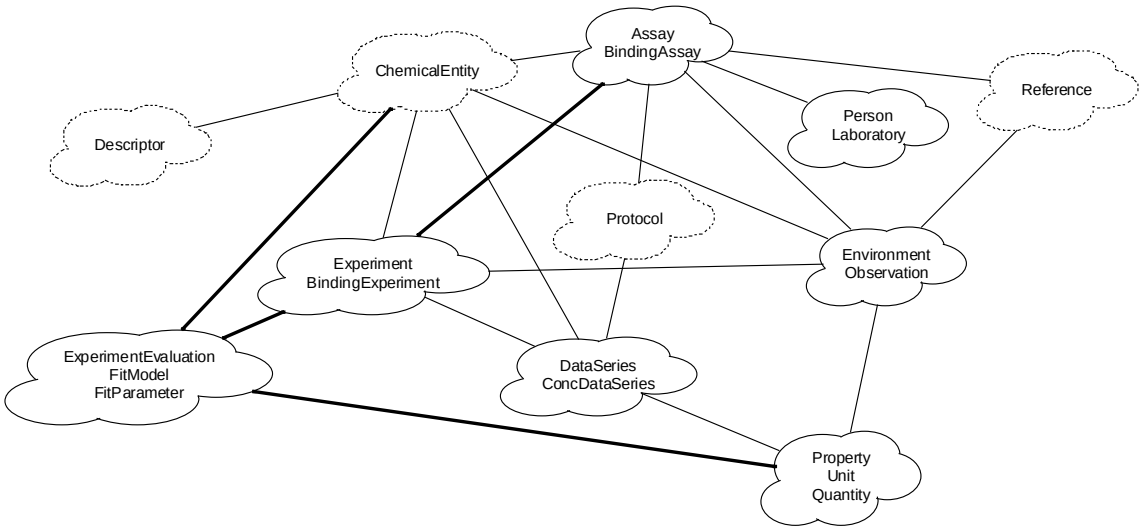


Figure 17. Division of the domain of interest into subdomains.

7.3 The Prototype for PAQS

The first stage prototype for a proteo-chemometric analysis and query system presented in this Thesis consists of the database schemas of chapter 8 implemented in Amos II. No graphical user interface or wrapper to external data sources have been implemented. A few examples of how data can be stored and queried are presented in Appendix B.

The scripts needed to generate the database, together with some sample data and further demonstration examples, are available as supplementary electronic material. The DBMS Amos II needed to run the scripts is available for download over the Web (http://www.dis.uu.se/~udbl/amos/ 2002-01-16).

8 Modelled and Implemented Subdomains

In the work for this Thesis many parts of the necessary database schema for a proteo-chemometric analysis and query system (PAQS) have been modelled and implemented in Amos II. This chapter describes and discusses these parts, including some background information and related work.

The database schemas described in this chapter all refer to the mediator of a mediator/wrapper architecture. I do not discuss the schemas of external data sources, although the presented schemas will in many cases provide good starting points for defining XML Schemas. In most cases the data will actually be stored in some external source, possibly on the Web, and the schemas are the representations in the integration schema of the mediator. Thus, data which is read from external sources can be viewed and manipulated by Amos II functions, presented to the user, and possibly stored in some local database. However, in the discussion of design solutions it is in many cases convenient to talk about attributes that "store" properties of a type, as if they were stored permanently in the mediator.

It has been difficult to choose the order in which to present the following subsections. The first section, which is quite long, deals with quantities and units, and after that a larger data structure called `DataSeries` is introduced. In the middle of the chapter the modelling of binding assays, binding experiments, and various supporting types is discussed. Finally, experiment evaluations (numerical fits) are discussed in a fairly long subsection. Each subsection should be fairly self-contained, with references to other sections.

The chapter contains a few code excerpts in AMOSQL and Java, where appropriate. Full code listings can be found in the electronic supplementary material. A short demonstration of the implementation is given in Appendix B.

8.1 Quantities and Units

There are many ways to model measurements. A few things in need of consideration is how to represent a measured value, how to represent the system of units, and how to handle error estimates of measured values. In the following subsections various approaches to representing a single measured value, units and errors will be discussed. The final schema which has been implemented in the prototype is given in section 8.1.10. It is worth noting that the database prototype uses this approach for single measurements, but another approach, presented in section 8.2, for (longer) series of data.

8.1.1 The Measured Value

First of all we need to determine how to represent the measured value, which is usually a real number. In a computer, real numbers are handled by some floating point number data type and in many programming languages one may choose a floating point type of appropriate precision. For example, Java has the relevant data types `float` and `double`, while Fortran77 has `real` and `double precision`. The DBMS Amos II has only one floating point data type, `real` (with approximately 15 significant digits).

For some properties we may know that the measured value is always an integer, e.g. the age of an (adult) person. Furthermore, some measurements do not result in a number at all but in a

choice from a set of predefined allowed values, an enumeration[207]. For many measurements in science a value is only interesting as part of one or several sequences of numbers (e.g. an infrared spectrum). In some occasions, the measured value may even be a complex number.

In the implemented prototype all measurements are treated as real numbers (using the Amos II literal data type `real`). Spectra can be handled as pairs of `DataSeries` objects, where each data series is a sequence of real numbers (section 8.2).

8.1.2 The Need for Units

In this section we will discuss a few alternative ways to represent systems of units. The approaches differ greatly in expressiveness, flexibility, and the amount of ”knowledge” that clients need to have to be able to use the units.

Why do we need units at all? In principle we could prescribe that all concentrations in the database system will be given in the unit nanomolar, all masses in gram, all volumes in millilitres, and so on. Alternatively, we could decide to always use the SI base and derived units[208]. Unfortunately, neither is convenient since there will typically be applications and user groups that insist on using their customary units. For example, the ”standard” unit of concentration in chemistry is mole/litre, also called molar. It would be very hard to convince chemists to give concentrations in the SI unit mole/meter^3. Choosing a system of units other than the SI system will eventually lead to the same kind of problem.

Using units is also more or less a prerequisite for interoperability. If we wish to use data from several data sources on the Web, we can be almost certain that not all of them use the same set of units. Obviously, we may wrap the data source to a common system of units, but as we will see there are several advantages of using explicit units.

Unit Conversion

If we have units in the database, we also want a means for converting between different units. This is very useful if we allow different units for the same physical dimension (e.g. meter, millimetre, and Angstrom (Å) for lengths), since we would then be able to add 0.0010 m and 1.0*10^7 Å to get 2.0 mm. Unit conversion will be necessary if we want to be able to access other data sources, even if we restrict our own database to have only one unit per dimension. The other data source will quite likely have its data in other units, and our wrapper will then need to convert from the external units to the internal units.

A few approaches to unit conversion will be discussed in the following sections. I have found very few publications on unit conversion. However, Novak (1995) discusses the conversion of units of measurements in the context of dimensional analysis (vide infra), and also describes algorithms[209] for conversion and simplification of units. Interestingly, he uses eight base units, the seven SI units and dollar.

---

[207] Furthermore, one often differentiates between ordinal variables (with some inherent relative order) and nominal variables (with no meaningful order) (Han and Kamber 2001, p 343). The colour of an athlete's eye would be a nominal variable, but the colour of the medal he or she won at the Olympic games would be an ordinal variable.

[208] There are seven SI base units (meter, kilogram, second, Ampere, Kelvin, mole, candela). The derived units are multiplicative combinations of the base units, e.g. meter/second and mole/meter^3.

[209] Of further interest is that Novak (1995) describes the implementation of the algorithms in LISP, one of the languages the Amos II system is implemented in. However, I have not used LISP for unit conversion, but AMOSQL and Java.

8.1.3 Dimensional Analysis

Why is it hard to represent units at all? One could think that it is just a technical matter, but actually there are quite a few intriguing problems in the conceptual modelling of units and measurements. This section will present a few of these problems, arising when dimensional analysis is used to relate units of different physical dimensions to each other. The database schema used for the prototype avoids the problems by not trying to keep these relationships at all.

Dimensional analysis is a technique which relies on the fact that "the various terms in a physical equation must have identical dimensional formulae if the equation is to be true for all consistent systems of units" (Pitt 1977). A mechanical quantity can be expressed mass, time and length, e.g. area = $(\text{length})^2$, energy = (mass) * $(\text{length})^2$ / $(\text{time})^2$, and so on. Science students early learn to use the technique for checking if the equations they use can be correct[210].

By dimensional analyses we know that we can only compare quantities that have the same physical dimensions. E.g., we may add a distance to a distance, but not a distance to a volume.

Some nice examples of the difficulties with dimensional analysis are the following[211]:

- A force multiplied by a distance may be either a torque or an amount of work done. Torque and work will have the same units (Newton*meter), but they are quite different physical properties, which usually would not make sense to add.
- On the other hand, it *does* make sense to compare a cup of salt with 100 gram of salt, even though the first is a volume and the second is a mass.
- Are the angles 40° and 400° the same? The answer depends on whether we deal with circular or rotational angle.
- The water solubility of a substance is usually given as the mass of substance it is possible to dissolve (at 25 °C) in 100 grams of water. Thus, if we subject this concentration unit (g/100g) to dimensional analysis we get a unitless dimension. On the other hand, mass%, volume%, and mole fraction are other dimensionless ways to measure concentrations. Clearly, these concentrations are not equivalent, but according to dimensional analysis we will be able to add quantities measured in these units. What is worse, we may add the interest on our bank account (1 % ?) to the concentration of carbon dioxide in normal dry air (0.03 volume%), but what does the result mean?

Thus, dimensional analysis is a powerful tool to relate and manipulate units (see also section 8.1.5), but there are some problems in constructing a general system without anomalies. This would be quite an interesting research problem for a computer scientist interested in modelling, and with a background in the natural sciences. It is, however, perhaps not of immediate concern to the PAQS project.

---

[210] For example, assume we want to calculate how far an athlete runs in 20 seconds if he has a speed of 10 m/s. We remember that speed (v), distance (d) and time (t) are related in some way, but how? We attempt the equation d = t/v. However, dimensional analyses tells us that the unit of [t/v] is second/(meter/second) = s^2/m, which is certainly not what we expected for a distance. So the equation we tried was wrong…

[211] The first three examples were taken from an 130-page unfinished work report (Kent, Janowski, Hamilton, and Hepner 1996).

8.1.4 Units as Character Strings, an Enumerated Type, or a Domain

Obviously, one simple way of representing units is as character strings. When a new quantity, e.g. 1.25 m/s, is entered into the database the user has to type in both ”1.25” and ”m/s”. The database administrator has no control over which units that are used, or that all users use the same character string when they mean one and the same unit. E.g., another user may write ”mps”, and some other will prefer to use ”cm/s”, or even ”inch/hour”. How shall the database system know that ”mps” and ”m/s” are the same, and how shall it know how to convert from ”cm/s” to ”m/s” if the units are simple character strings?

In the programming language C an enumerated type can be used to define which values an attribute may take[212]. Similarly, ODMG's object model (3.2.3) has the literal data type `enum`[213]. For relational databases we instead speak of "domains", and an attribute's domain is the set of values that can be assigned to the attribute[214].

To use an enumerated type or domain solves the problem of synonyms, but not conversion between different units[215].

8.1.5 Units by Vectors

Hamilton (1996) desribes an approach for handling units which relies on vectors and dimensional analysis (see section 8.1.3). One first defines a set of base units, e.g. the seven SI base units. Then all other units can be derived from these. Assume that we have defined only two base units: metre (for length), and second (for time). These units span up a "unit space", and we can represent the unit of the property length by the vector < 1, 0 >, and the unit for acceleration (m/s^2) by < 1, -2 >. Hamilton suggests this approach as a means to represent units in a compact way, with minimal agreement among communicating parties.

The vector approach is very expressive since the relationships between different physical properties are preserved. The approach is compact since it doesn't rely on a large enumeration of units. The disadvantage that there is only one unit for each dimension remains. More serious disadvantages are that the method has problems with handling dimensionless properties, and different properties having the same dimension (Hamilton 1996).

8.1.6 Units and Quantities as Objects

A quite different approach to that of the previous section, and one perhaps more suitable in an object-oriented environment, is to represent both units and measurement values as first-class objects.

---

[212] In an object-oriented language, e.g. Java, enumerated types may be substituted by the "typesafe enum pattern" (Bloch 2001).

[213] However, the Java mappings for `enum` are not yet defined.

[214] In ISO SQL 2 a domain is a name, a data type, an optional default value, and an optional CHECK statement (Connolly and Begg 2002). An example is `CREATE DOMAIN Gender AS CHAR DEFAULT 'F' CHECK (VALUE IN ('M','F'));`, whereupon an attribute `sex` could have the type `Gender`. However, according to O'Neil and O'Neil (2001, p 419) none of the major DBMS products support such an enumerated data type. Instead they suggest the solution with an additional table containing all allowed strings.

[215] A table with all valid conversions could be used. However, it is not trivial to keep such a table up-to-date as new units are added.

Fowler (1997) suggests a schema similar to that in Figure 18 for representing patients in a hospital, with `Quantity` as a type that knows both the value (`amount`) and the unit of a measurement.

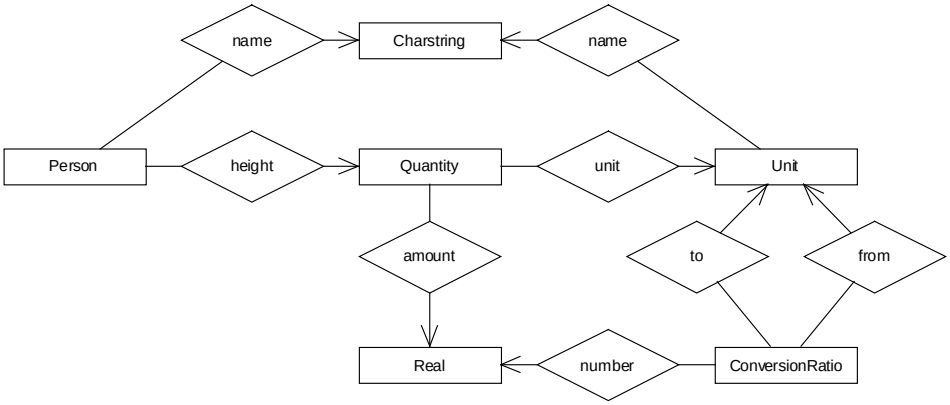


Figure 18. Quantity and Unit as concepts for representing measurement values (after Fowler (1997)[216]).

Fowler also advocates the use of compound units. The left part of Figure 19 uses bags, while the right part only uses sets. (Both approaches would work in Amos II.) Fowler's approach is very similar to the vector approach presented in section 8.1.5, but object-oriented.

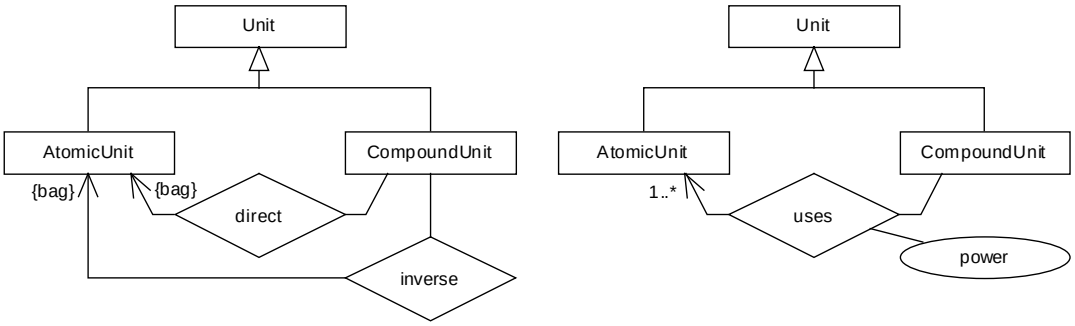


Figure 19. Two ways of modelling compound units according to Fowler[217].
(After Fowler (1997), adapted to the functional data model notation of the present thesis.)

It seems that compound units and conversion ratios may be combined, as in Figure 20[218]. For example, if we have `:meter` defined as a base unit for length, we could proceed according to the following[219] to construct the concentration unit `:nanomolar`:

---

[216] Figure 18 has been drawn in the "functional EER" model language first encountered in section 3.2.5. The original Figures of Fowler are drawn in "crow-feet notation", and on his home page (Janicijevic and Fowler 2001) they are available in UML. For clarity the attributes `name`, `height`, and `amount` are drawn as explicit functions from user-defined types to literal data types.

[217] In by-passing, we may note that ”AtomicUnit” is not a suitable name in a database for the natural sciences. Atomic units is a system of units often used in quantum mechanics and particle physics instead of the metric system!

[218] The solution in Figure 20 closely resembles the "Composite" design pattern (Gamma et al 1995; Grand 1998). However, that pattern is used to build whole-part hierarchies, with no sharing of "parts" between "wholes". In contrast, a unit (e.g. metre) could potentially be used by many derived units (e.g. m^2, m/s, dm).

[219] 1 dm^3 = (1 m * 0.1 dm/m)^3 = 0.001 m^3; 1 molar = 1 M = 1 mol/dm^3; 1 nM = 10^{-9} M. We choose to consistently first apply `conversionRatio` and then `power`.

```
uses(:decimeter) = < :meter, 1, 0.1 >
uses(:dm3) = < :meter, 3, 0.1 >
uses(:molar) = bag( < :mol, 1, 1 >, < :dm3, -1, 1>)
uses(:nanomolar) = < :molar, 1, 1.0e-9 >
```

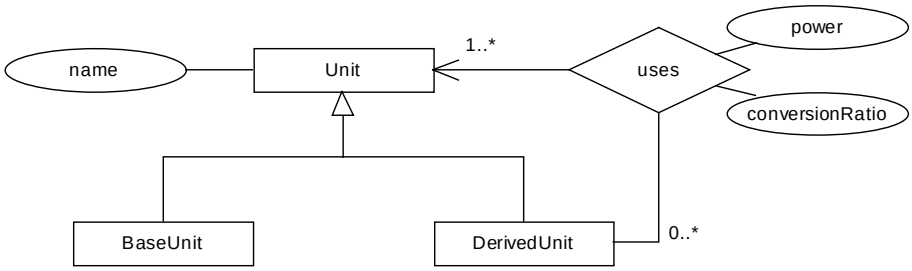


Figure 20. A schema for units, including conversion and compound units.

Note that in Figure 20 the function `uses` maps `DerivedUnit` onto the ”abstract” type `Unit`, not onto `BaseUnit`. I.e., we may define a derived unit in terms of other derived units. During this definition care must be taken that no circular paths are constructed, and it will be more work to follow the path from a derived unit all the way to its constituent base units than in Figure 19. However, an advantage is that it is easy to define a new derived unit, even though it may be "far away" from the base units (cf. `:nanomolar` in the example above).

In the prototype all units are treated equally, no difference is made between base units and derived units. One of the schemas above should be implemented as soon as we want the database to do more general calculations than unit conversions.

An example of when we need a schema with compound units is when we want to be able to do arithmetics with quantities for different properties. It is easy to construct a method for adding two quantities of the same physical property, e.g. adding (15 cm) to (1.2 dm), and this is done for the prototype. However, if we wish to be able to multiply two quantities and let the database return an answer with correct unit we need compound units:

```
:total_amount = <0.00102,:mol> + ( <4.02,:gram> / <432.3,:gram_per_mol> )
```

8.1.7 Accuracies and Error Estimates

In science the accuracy of the recorded value is often important. For example, there is a significant difference if we say that the concentration of a solution is 0.50 nanomolar or 0.500 nanomolar (nM). In the latter case we know the concentration of the solution to a ten-fold higher accuracy[220]. Even better[221] is of course if we explicitly include an error estimate and give the concentration as (0.500 ± 0.007) nM. To further complicate things it is not uncommon that the estimated error is asymmetric, e.g. $0.500\ ^{+0.007}_{-0.003}$ nM. These considerations are even more important in statistical data analysis, where we could need to describe not only the mean of a distribution, but also its variance, skew, and kurtosis.

Once we have introduced the type `Quantity` (in some way similar to Figure 18) it is straightforward to represent error estimates of individual measurement values. Easiest is to add a real attribute `errorEstimate` to the data type `Quantity`, with the understood assumption that `amount` and `errorEstimate` are given in the same unit. This has the

[220] The quantity 0.50 nM is given with two significant digits, and 0.500 nM with three.

[221] Some physicists claim that having a measured value without an error estimate is no more worth than having no measured value at all.

disadvantage that it does not tell *how* the error was estimated, i.e. if it is a known constant of the measurement process, the variance of a series of measurements, or something else.

A bit more elaborate would be to create a separate data type `ErrorType`, to describe error estimates more generally. With the schema of Figure 21 it is even possible to have several error estimates for a single measurement value.

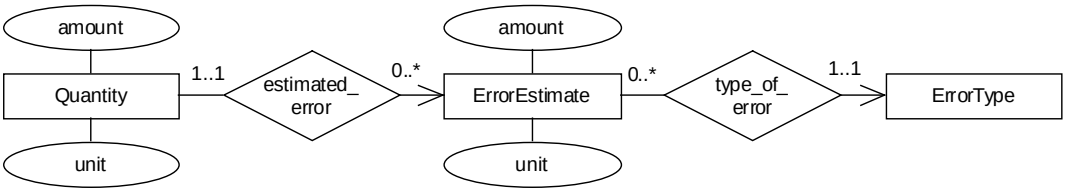


Figure 21. A schema for measurement errors.

No error estimates have been implemented in the prototype database, but the schema in Figure 21 would be easy to implement once the properties of the type `ErrorType` have been established. Note, however, that it is one thing to implement a schema to tag each quantity with one or several error estimates, but it is quite another thing to use these errors in arithmetic calculations[222].

This is an area worth more work. We would indeed like a proteo-chemometric database to be aware of accuracies, if not error types. Thus, the addition of 0.50 gram and 0.3012 gram should result in 0.80 gram (not 0.8012 gram!). Similarly, adding 1.0 microgram to 1.0 kilogram should result in 1.0 kilogram (not 1.000001 kilogram!).

Finally, we may note that the way Amos II displays real numbers have no relation to the way we store them in the database. For example, 1.0e6 and 1.0e5 will be displayed as `1E+06` and `100000.0`, respectively. This is worth remembering, and with a graphical user interface, e.g. written in Java, the programmer will have better control of how numbers are displayed.

8.1.8 Properties and Unit Types

A subject which Fowler (1996) does not discuss is how we can know which physical property the unit is used for. If we have an attribute `height(Person)` it is easy to understand what the quantity 1.80 meter means. Similarly, if a quantity is associated with an observation, which in turn is associated with a property, the meaning of the quantity is clear. But these are both examples of data already stored in the database. When the user shall insert a quantity in the database, how do we prescribe which units that are suitable for the property mass?

A relatively simple solution is to put a relationship between the types `Unit` and `Property`, in Amos II as a function `property(Unit)->Property`. As long as we don't have many properties in the database this will work fine, and probably the approach only gets into problems when there are several properties having the same set of units[223]. Since this could be a problem, e.g. when it comes to different properties using molar, molar^{-1}, and s^{-1}, we introduce a type `UnitType`, see Figure 22.

[222] Statistics may tell us how to add two values with standard deviations to give a new value with standard deviation, but to add one quantity with one error type to another quantity with another error type is considerably harder, and probably impossible to do in a general way.

[223] E.g., the physical properties work and torque both have the SI unit Nm (Newton meter).

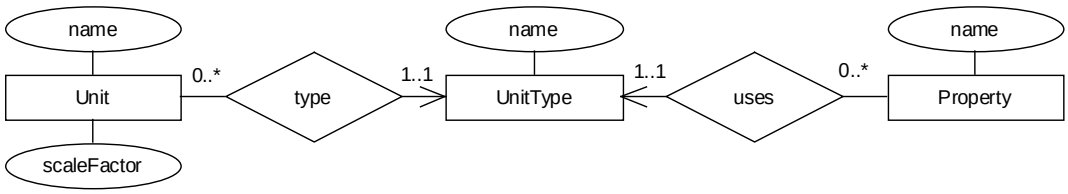


Figure 22. A schema for grouping sets of units, and for coupling these to properties.

With this solution we get great flexibility, it is easy to keep track of which units that ”belong together” and can be converted to each other. The scale factor is a number used for converting from one unit to another[224]. As an example, assume we wish to convert the quantity 9.5 dm to centimetres. First of all, both the units `:decimeter` and `:centimeter` belong to the UnitType `:length`, so we may converted between them. Next we find that `scaleFactor(:decimeter) = 0.1` and `scaleFactor(:centimeter) = 0.01`. The correct new value is 9.5*(0.1/0.01) = 95. For each `UnitType` there will be one ”base unit” with scale factor 1.0, typically one of the SI base or derived units. (For concentration the unit molar has scale factor 1.0.)

8.1.9 Concentrations

Since the database prototype is developed for the domain of proteo-chemometrics, concentration is an important concept. A problem is that there are many different meanings of the term ”concentration”. What we have used so far is also called ”molarity”, from the unit molar (mol/dm^3). I.e., the molarity of a solution of solute A in solvent S is the amount of A divided by the volume of the solution. In physical chemistry it is often convenient to work with ”molality” (unit: molal (mol/kg)), which is the amount of solute A divided by the mass of solvent S. Molality is also a concentration, and many other exist (e.g. mole fraction, mass percent, volume percent[225]). If we are dealing with dilute water solutions the molarity and the molality are approximately equal, but this is not generally valid. Thus, there is no single conversion factor between molarity and molality[226].

In Figure 22 we have constrained each `Property` to have a single `UnitType`. This enables us to ask questions of the kind ”which is the base unit for the property dipole moment?”. Thus, we cannot have two `UnitType` objects for concentration. Instead we make the choice that concentrations in the prototype are given in some unit of type molarity[227].

For the property solubility the same type of problem arises as for concentration. Some data sources give the solubility of substance A in solvent S (typically water) as the mass of A that dissolves in 100 gram of S (at a specified temperature). Other data sources instead takes the mass of A that dissolves in 100 millilitre S. Since solubilities are usually only given with one or two significant digits, the two numbers will in practise be the same for the solvent water[228].

[224] Note that Figure 22 assumes that scale factors are exact. There is no error estimate associated with a scale factor.

[225] See any textbook on general chemistry, e.g. (Silberberg 1996) for a more detailed explanation.

[226] Actually there is not a single conversion factor between molarity and molality even for a given solution. The problem is that the molarity depends (to a very small degree) on the temperature, while the molality does not. However, a discussion of this leads us far into the subject of physical chemistry.

[227] I.e., in Amos II we have the following stored function: `uses(:concentration) -> :molarity`, where `:molarity` is an object of type `UnitType`, and `:concentration` is an object of type `Property`.

[228] The density of water is (with two significant digits) 1.0 g/ml from 5 °C to 30 °C.

8.1.10 A Database Schema for Quantities and Units

Figure 23 shows the database schema for quantities and units that has been implemented in the prototype. The type `Observation` is described further in section 8.3[229]. Note that some of the features discussed above have not been implemented, e.g. error estimates and the concept of compound units.

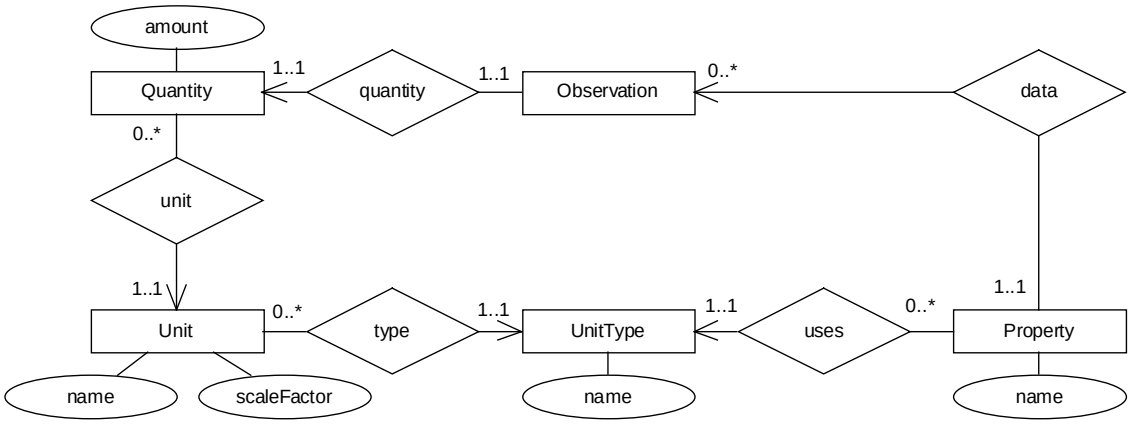


Figure 23. Implemented database schema for quantities and units.

In the schema diagram of Figure 23 several types have an attribute `name`. These attributes are implemented as keys in the Amos II prototype. E.g., the name of a unit is unique – there can be no two units with the same name.

An implementation according to the schema of Figure 23 has the following advantages:

- It can handle conversion between units belonging to the same `UnitType`,
- it is easy to extend to new units (each unit is an instance of the type `Unit`), and
- it can handle additions and subtractions of quantities given in different units (as long as the units belong to the same `UnitType`).

However, the schema does *not* keep relationships between dimensions, to accomplish this some variant of compound units need be implemented. The representation of units is not particularly compact (see Hamilton 1996).

8.1.11 The Quantity Type Revisited

The type `Quantity` is implemented in AMOSQL with the following interface[230]:

```
type
    Quantity
```

[229] For example, this type will tell us *what* it is we observe (The concentration of which ligand? The volume of which assay?), and *how* we have observed it (with apparatus X, data from reference Y).

[230] This and other interfaces listed in the Thesis are not complete. Only the (subjectively) most important or interesting methods and functions are included. ”Methods” are functions mapping objects of type `Quantity` onto a single type, while "functions" are Amos II functions with several arguments or tuple results. (The division into methods and functions is taken over from goovi , a graphical browser for the Amos II system (Cassel and Risch 2001).) Methods and functions may both be Amos II stored or derived functions, e.g. `amount(Quantity)` is a stored function, but `toString(Quantity)` is a derived function. ”Constructors” are functions intended for object creation, similar to constructors in Java and other object-oriented languages.

```
methods
    amount( Quantity q key ) -> real [stored]
    unit( Quantity q key ) -> Unit [stored]
    toString( Quantity q ) -> charstring

functions
                /* unit conversion */
    convert( Quantity q, Unit u ) -> real
    convert( Quantity q, charstring unitName ) -> real

                /* arithmetics, with choice of result unit */
    plus( Quantity q1, Quantity q2, Unit resultUnit ) -> Quantity
    minus( Quantity q1, Quantity q2, Unit resultUnit ) -> Quantity

                /* arithmetics, result given in unit of q1 */
    plus( Quantity q1, Quantity q2 ) -> Quantity
    minus( Quantity q1, Quantity q2 ) -> Quantity

                /* equality of Quantity objects */
    equal( Quantity q1, Quantity q2 ) -> boolean

                /* comparisons */
    lessThan( Quantity q1, Quantity q2 ) -> boolean
    greaterThan( Quantity q1, Quantity q2 ) -> boolean
    lessThanOrEqualTo( Quantity q1, Quantity q2) -> boolean
    greaterThanOrEqualTo( Quantity q1, Quantity q2 ) -> boolean

constructors
    createQuantity( real amount, Unit u) -> Quantity
```

A few of the functions will be demonstrated in Appendix B[231]. Note that the second pair of functions `plus` and `minus` in the listing are overloaded on the corresponding native Amos II functions (which take numbers or charstrings as arguments). We could also overload `abs`, `max`, and `min`, but without a framework for compound units we cannot implement `times` or `div`. Finally, it could be useful to overload a few aggregation functions, e.g. `sum` and `average`.

The interfaces of `Unit` and `UnitType` are not listed here, but one important point is that `Unit` objects should be created with the ”constructor”

```
createUnit(charstring unitName,real scaleFactor,UnitType ut) -> Unit
```

which assures that the database is not populated withv equivalent units. The function `createUnit` checks that the unit does not already exists under the same or another name (but the same unit type and scale factor). On the other hand, since name is a key for `Unit` the Amos II system will throw an error if we try to create a unit with name “m” for molality if the name “m” already is in use for meters.

8.2 Data Series

In many cases it is not a single measurement that is interesting, but instead a series of data points, all referring to the same property. Typical examples from science are different kind of spectra. For example, an infrared spectrum may record the percentage of infrared light transmitted through a sample for a large number of different frequencies[232]. Other examples

[231] `equalTo(Quantity,Quantity)->boolean` would be more in line with the naming conventions of the Java collections framework. However, this caused a strange overload error in some of the test scripts, probably due to a bug in Amos II.

[232] IR spectra are usually measured in % transmitted light intensity versus wavenumber (frequency divided by speed of light).

are time series, e.g. how stock exchange rates change over a day or over a year. For a competition experiment in a binding assay, the concentration of bound radioligand varies with the concentration of competitor ligand.

One thing these examples have in common is that one particular point taken alone is of limited interest, and that it is often useful to plot the data in an xy-diagram. We need to see quite a range of an infrared spectrum in order to identify the chemical compound, we need to see the trends (and understand them) in order to make money on the stock exchange, and for the competition binding experiment we need a series of points to make a reasonable fit to some model of the binding interactions.

In a data series each data point contains the same kind of data. Such data series will typically be stored as columns in an Excel sheet (see Figure 24). There are several alternative ways to represent such data series. Which is most appropriate will depend on how strong the coupling is between the data points, on the data model of the DBMS, and on the kind of applications to use the data. In Amos II, a suitable data type to use is `vector`, which makes it possible to represent an ordered sequence of numbers (or any objects).

8.2.1 Dependent and Independent Variables

Typically, the kind of data described above have one independent variable and one dependent variable ("x and y"). For the competition binding experiment the amount of radioligand bound to receptors at cell membranes depend on how much of competing ligands is present in the assay.

It is also possible to consider situations with several dependent variables (a meteorologist measuring temperature, atmospheric pressure, humidity et cetera at a series of altitudes) and situations with several independent variables (the same meteorologist measuring the temperature for a series of points in space (different longitude, latitude, and altitude).

Binding experiments usually have only one dependent and one independent variable, i.e. a single radioligand is present in the assay, and the concentration of a single (labelled or non-labelled) ligand is varied. However, the prototype described in this Thesis has been designed to accommodate several dependent and independent variables. This decision was taken in order to accommodate for other types of experiments, and also possible new future binding experiment designs.

An Example of a Series of Measurements

Let us consider a competition binding experiment. In principle we have one independent variable, the concentration of the competitor (in BindAid called ”varying”), and one dependent variable, the concentration of bound radioligand (called ”bound”). In practice, we will often have two or more ”parallel” experiments in order to get better statistics. Each of these parallel experiments have the same experimental set-up, and the same series of varying ligand concentrations (the same independent variable). However, the values in the series of bound ligand concentrations may differ slightly due to statistical errors[233].

Figure 24 shows the beginning of such a data series, taken from a spreadsheet. Here we have two parallel experiments. There are three columns of raw data (”varying”, 2* ”DPM”) and one column of derived data (”bound” = average(DPM) * conversion factor).

[233] If one of the data series has a systematic error it was in fact not performed in exactly the same way as the others, and hence they were actually not parallel. Thus it is important to have the same experimental procedure, and experimentator, for parallel experiments.

No.	Varying	Bound	DPM	DPM
1	5000	0.01125	140	130
2	1000	0.01104	135	130
3	200	0.02000	345	135
4	40	0.01542	190	180

Figure 24. Example data from competition binding experiment. From the BindAid manual (Wikberg).

There is a choice as to whether we store raw data (counts per minute, or disintegrations per minute) or calculated concentrations for bound ligand concentrations. From a database perspective it would not be customary to store both of them since a simple factor can be used to convert between the two. One objective with the database system to be developed is that it should be able to analyse the same data over and over again. Hence, I have provided the possibility to store the raw DPM data in the prototype system. Of course, it is equally possible to store averaged concentrations, but both should not be stored. In the prototype it is possible to take raw data (one or several DPM columns in the spreadsheet) and by a simple function invocation convert them to an averaged concentration data series in units of nanomolar (nM). This averaged series will not be stored, but calculated as needed.

In practise, experimental raw data will not be stored in the main-memory mediator, but in some data source on disk, e.g. in a relational database or as XML files. In this case the translator which imports the data could be designed to either convert all concentration data series to the units nanomolar, or keep raw data.

8.2.2 Representing a Series of Measurements

A solution often encountered in science is to represent the kind of data series shown in Figure 24 as a sequence of (x,y) points, or (x, y_1, y_2,…, y_m) points if we wish to have *m* dependent variables in the same experiment (2 for Figure 24). A data series would thus be represented by a vector of (*m*+1)-tuples.

Measurement Points

If we instead use an object-oriented approach, we get a schema diagram as in Figure 25. Each row now corresponds to a `MeasurementPoint`. We constrain all these measurement points for the bound series to have the same unit, and those for the varying series to have the same unit.

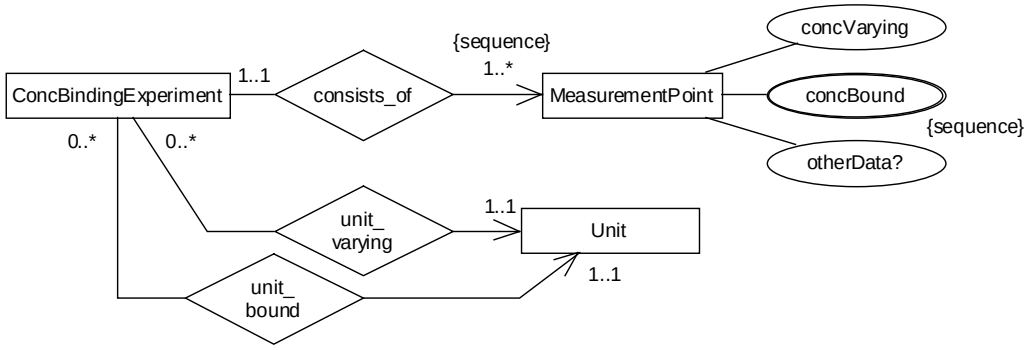


Figure 25. The `MeasurementPoint` approach.

The schema above is valid if there is only one varying concentration per Experiment. The double rings around `concBound` mean that `concBound` is a multivalued attribute of `MeasurementPoint`, i.e. that we may store values from parallel experiments.

Data Series

We now turn to another approach, to let each *column* of Figure 24 correspond to a `DataSeries`. For example, the column named ”varying” in Figure 24 will correspond to one `DataSeries` object, and this object will have an attribute `values` containing four real numbers in a specified order. In Amos II we implement this as a `vector` of real numbers. Obviously, one constraint we need to take care to implement is that all `DataSeries` belonging to an `Experiment` should have equally long `values`-vectors.

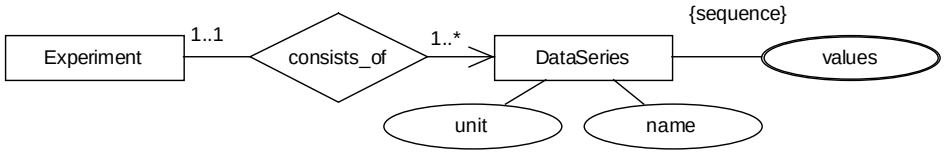


Figure 26. The `DataSeries` approach.

The schema presented above fails to work nicely if one `DataSeries` has missing points, if the unit is changed in the middle of the series (from nM to μM), or if some “protocol variable” is changed (e.g. detector amplification). For now, we assume the latter two things never happen[234].

Points That are Not Valid

The “missing point problem” is closely connected to the situation when the experimentator knows that one point is invalid (due to some experimental mistake). For example, suppose we want to average the two data series {34.0, 35.0, 38.0, 50.0, ...} and {36.0, 32.0, 523.0, 45.0,…} and that it is known that point 3 in the second data series is (for some reason) incorrect. Then it should be possible to mark exactly this value as “non-valid”, so that it is not used in the averaging.

Thus, we may include a Boolean flag `valid`. (If a data point is missing we may arbitrarily put the value to zero and mark the point invalid.) This approach could also be of advantage if we would like to be able to store error estimates of individual values[235]. Similarly, we could label each point with a comment (text), see Figure 27.

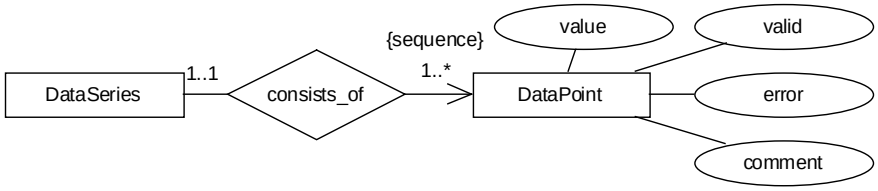


Figure 27. Explicit `DataPoint` objects, with boolean `valid` flags.

8.2.3 How Data Series are Implemented in the Prototype

The solution chosen for the prototype is a hybrid of the last two approaches. We will handle the data point values as sequences of real numbers (`values`), and indicate which (if any) points that are non-valid by including their indexes in the set `nonvalids`.

[234] In practise, such a situation would be handled by splitting the data into two distinct `DataSeries` objects, one with detector setting A and one with detector setting B. The person evaluating the experiments can of course still make the decision to analyse both these data series together. Depending on the application using the database it could instead be more convenient to split the data into two distinct `Experiment` objects.

[235] I.e., this is necessary if we have some direct measure of the error, not just a statistical model.

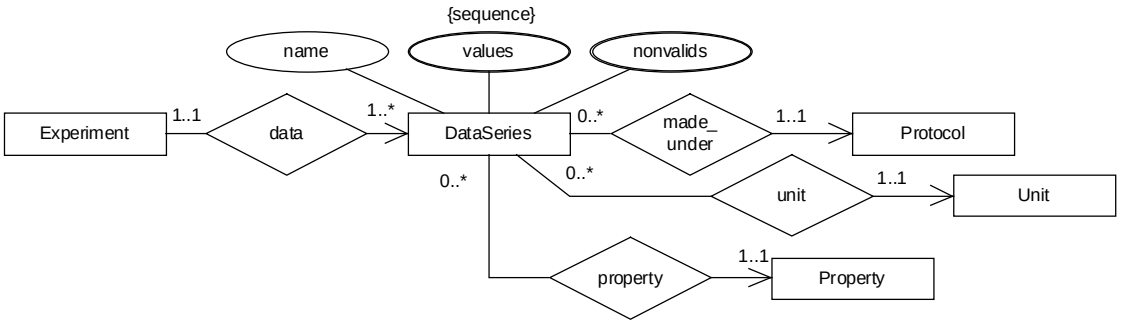


Figure 28. The schema for `DataSeries` implemented in the prototype database.

A "normal" `Experiment` object would be linked to two `DataSeries` objects, one for the independent variable, and one for the dependent.

In Figure 28 we have introduced relationships from the type `DataSeries` to the types `Property` and `Protocol`. The former of course indicates which physical property the data series records (e.g. time), and the latter is a specification of the experimental set-up. For data series of bound radioligand concentrations we could for example record information about which beta-counter detector that was used. This kind of information is obviously possible to get only from some laboratories.

Nonvalid Data Points

In the Amos II prototype of PAQS the attribute `nonvalids` is implemented as a *bag*[236]. Obviously a data point should be valid or non-valid, not doubly non-valid, so a *set* would be the appropriate data structure. The work-around in the prototype is to use two functions `invalidateDataPoint( DataSeries, integer )` and `validateDataPoint(DataSeries, integer)` that should be used instead of directly invoking AMOSQL commands (e.g. `invalidateDataPoint(:ds,3);` instead of `add nonvalids(:ds)=3;`). These two functions check array bounds, so that we do not store an index in `nonvalids` to an element that does not exist in the vector `values`. Furthermore, `invalidateDataPoint` only accepts valid data points, effectively making `nonvalids` a set as long as no elements are added to the bag directly.

The dependency (or coupling) between various attributes to a Data Series object is rather strong. Some precautions have been made to avoid problems. For example, the function `removeDataPoint(DataSeries, integer)` will not only remove a data point from the vector `values`, but also try to remove the point from the bag `nonvalids`. Furthermore, those integers in `nonvalids` that are larger than the index of the removed data point will be decremented in order to reflect the disappearance of the data point.

However, the prototype is still full of “loopholes”. E.g., it is perfectly possible to first define a data series with 10 values, then invalidate point number 9, and then reassign `values` to another vector (say of length 7). If this reassignment is made by executing an AMOSQL command, no automatic concomitant reassignment if the bag `nonvalids` will be performed.

[236] A mathematical bag is an unordered collection of objects. In contrary to a mathematical set, a bag may contain duplicates.

8.2.4 Concentration Data Series

Each data series representing a varying ligand concentration clearly needs to be associated with the corresponding chemical entity in some way. A further peculiarity with concentration data is that we may wish to store "raw" detector readings from some detector which does *not* display concentration units. In the BindAid manual (Wikberg 2001) this is the case for all radioligand data, where concentrations are entered as beta-counter readings (in the units "disintegrations per minute") and then converted to nanoMolar.

We could say that a special thing for a data series representing beta-counter readouts or other "raw" concentration data is that there should be a way to convert the values to nanomolar concentration units. We could either consider a "DPMConcDataSeries" subtype of `ConcDataSeries` (left part of Figure 29), or a "RadioDataSeries" (to the right in Figure 29). Other kinds of data series do not need to be subtypes of `DataSeries`. E.g. a time data series from a time binding assay is simply a sequence of real values measured in some unit of time.

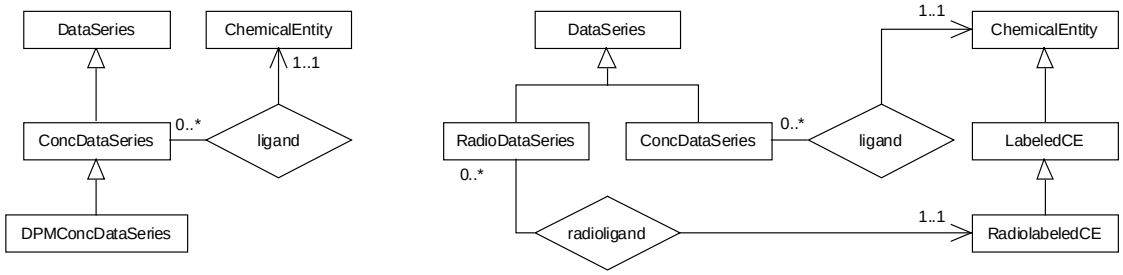


Figure 29. Two alternative ways of representing data series for concentrations.

The solution chosen for the prototype is to subtype `DataSeries` to `ConcDataSeries`, but not further. We treat beta-counters simply as a special kind of concentration detectors. Since we know the conversion factor from DPM (disintegrations per minute) to nM (nanomolar) a "DPMConcDataSeries" can be handled just as a `ConcDataSeries`. We simply introduce an new unit of concentration for each conversion factor[237].

One disadvantage by the simplification made here is that we loose some of the domain semantics. We know that a "RadioDataSeries" must pertain to a radiolabelled chemical entity, but if we have only `ConcDataSeries`, we cannot constrain this to be true. I believe there is still an advantage to keep this part of the schema relatively simple and flexible. Therefore, I have chosen to not make any other subtypes of `DataSeries` than `ConcDataSeries`.

[237] Appendix 1 of the BindAid manual (Wikberg 2001) describes how to calculate a conversion factor from dpm to nM: f = 2.22 * a * V, where a is the specific activity of the ligand (in the unit Curie/mmol), and V the assay volume (in µl). The conversion factor so obtained has the unit dpm/nM, and by dividing a "dpm value" by the factor we get the corresponding "nM value". Note that the conversion factors implemented in the prototype are defined in the opposite way: By multiplication with a factor 0.001 we turn a length measured in meters into millimetres.

If we know the conversion factor in the BindAid style it is easy to calculate a scale factor to be used in the schema of units described in Section 8.1.6, as long as we remember that the base unit for concentration is molar, not nanomolar. For a conversion factor of 13486 nM/dpm we get the scale factor as `times(div(1.0, 13486.0), 1.0e-9);`

8.2.5 Averaging and Conversion of Data Series

For a database of scientific data it is important to have good primitives for manipulating data. For single numbers, Amos II has the usual arithmetic operations, and as discussed in section 8.1.11 the prototype is implemented with functions for addition and subtraction of quantities belonging to the same unit type (but not necessarily the same unit).

When we deal with data series in the form of Amos II vectors we obviously want some means for vector manipulation too. The native vector implementation in Amos II allows the user to retrieve an element from the vector (e.g. `values(:myDataSeries)[2]` to get the third element), but it is not possible to easily reassign vector elements. In the prototype, the function `update(DataSeries ds, integer idx, real new)` accomplishes this, but the implementation in AMOSQL is not very efficient (a repeated concatenation of vector elements).

A few vector operations have been implemented as foreign functions by methods in the Java class `Prot1Vector` (see Appendix C). These methods are used when a data series is converted to a new unit (method `Prot1Vector.scaleVector`), and in a few other places. I believe additional operations should be implemented as part of the core Amos II system in order to get a more full-feathered scientific database engine. For larger applications the speed of these operations will be important.

An example of when it is useful to be able to average a set of data series is when an experiment has two parallel "DPM" data series of beta counter read-outs, and we want to convert them to a single data series in units nanomolar. The `values` vector for a new (averaged) data series may be obtained with the function `average_dataseries(bag b, Unit u) -> vector of real`. Some useful utility functions which make the averaging easier have been implemented:

- the function `average2nM(bag) -> vector of real` which takes a bag of `DataSeries` as argument and calculates their average in nM units,
- `average_bound(BindingExperiment,ChemicalEntity) -> vector of real` which averages (in nM) all "bound" data series for a specified `BindingExperiment` and `ChemicalEntity`, and
- `average_bound(BindingExperiment) -> <charstring, vector of real>` which averages (also in nM) all "bound" data series for each distinct ligand in a specified `BindingExperiment`.

The implementation of these functions is described in more detail in Appendix C.3.

8.2.6 Interface of Type DataSeries

The interfaces of the types `DataSeries` and `ConcDataSeries` are available as supplementary material. Some useful functions, partly mentioned above, are listed below. By using the constructors we ascertain that the vectors `values` is initialised to an empty vector, {}, and not NIL.

```
functions
    isNonValid( DataSeries, integer index ) -> boolean
    isValid( DataSeries, integer index ) -> boolean
    invalidateDataPoint( DataSeries, integer index ) -> boolean
    validateDataPoint( DataSeries, integer index ) -> boolean
    addDataPoint( DataSeries, real value ) -> boolean
    update( DataSeries, integer index, real new_value ) -> boolean
```

```
        removeDataPoint( DataSeries, integer index ) -> boolean
        convert_series( DataSeries, Unit ) -> vector of real

    constructors
        createDataSeries( charstring name ) -> DataSeries
        createConcDataSeries( ChemicalEntity ligand ) -> ConcDataSeries
```

8.3 Observations, Properties, and Environments

In this section we introduce the types `Environment` and `Observation`. They both take their origin in measurements of physical properties, either taken from the literature, or performed by the submitting laboratory. In section 8.1 we saw how measured values are represented as objects of the type `Quantity`, where each quantity has an "amount" and a unit. The type `Observation` is used to link a quantity to a property (*what* was measured), a protocol (*how* the measurement was made), and a reference (*who* made the measurement and *where* has it been published).

8.3.1 Observations are Data for Properties

In Figure 30 we let a function `data(Property) -> Observation` relate properties and observations. The alternative is to have a function `made_for(Observation) -> Property`[238]. Although the choice in Figure 30 might seem odd for properties such as temperature and pressure (having no associated chemical entities), it is quite convenient when we deal with binding affinities (an "intensive bimolecular property", vide infra)[239].

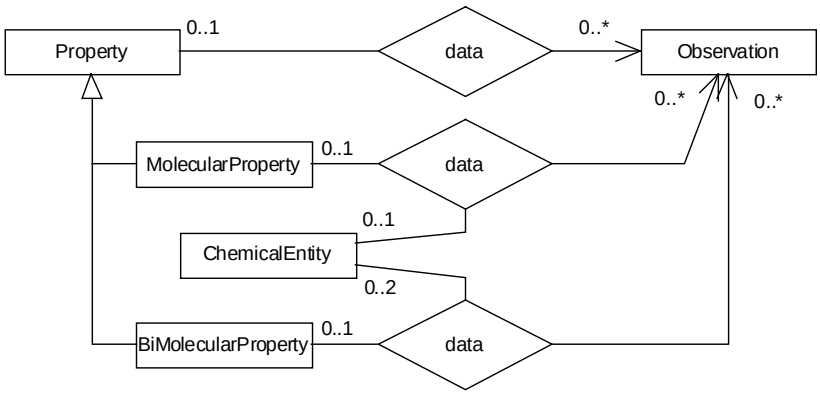


Figure 30. Observations and properties.

In Figure 30 we have subtyped `Property` to `MolecularProperty` and `BiMolecular-Property`. Here we make a difference between intensive and extensive properties (vide infra), and we only include *intensive* properties as objects of these two subtypes.

[238] If that approach had been chosen the title of this section would have been: "Observations are Made for Properties".

[239] Obviously, it must be possible to navigate easily in both directions. This is true with the query language AMOSQL, but to make the navigation even easier, both the functions discussed were implemented (`data` as a stored function, and `made_for` as a derived function).

Interface of Observation

The interface of the implemented type `Observation` is available as supplementary material, but a few functions for facilitating access of stored data are worth mentioning here. There are three stored functions corresponding to the three `data` relationships in Figure 30 :

```
data( Property nonkey ) -> Observation key
data( MolecularProperty nonkey, ChemicalEntity ) -> Observation key
data( BiMolecularProperty nonkey, ChemicalEntity, ChemicalEntity )
-> Observation key
```

The derived functions `made_for(Observation) -> Property` and `made_for_entity( Observation ) -> bag of ChemicalEntity` may be used to get the relevant property and one/two chemical entities, respectively. Finally, there is a whole range of functions `allObs`, with different arguments, each returning a bag of observations as the result of an AMOSQL query. E.g., `allObs(:affinity)` would return all affinity observations, while `allObs( :affinity, :alfa2A)` would return all affinities including the chemical entity `:alfa2A` .

Intensive and Extensive Properties

Intensive properties are those properties that do not depend on how much of the sample we have, e.g. pressure, temperature, concentration, and molecular weight. Extensive properties are mass, volume, energy content et cetera. If we take a sample of 1 dm^3 water and divide it into two beakers containing 0.5 dm^3 each, the temperature in both beakers are the same as in the original sample, but the volume in each beaker is less than the original sample volume.

The molecular properties we store are intensive (e.g. dipole moment). Extensive properties (e.g. mass) certainly may belong to a specific ChemicalEntity, but they also belong to a sample or a some preparation procedure, which we have not yet tried to model. It is likely that a convenient way to model a sample preparation[240] will include observations linking intensive properties to chemical entities.

8.3.2 Problems with the Property Hierarchy

Some more work is needed on modelling properties. The name "MolecularProperty" suggests properties of the kind molecular weight, dipole moment, solubility et cetera, i.e. properties one expects to find in a tabulation of chemical data for different substances. This was also the original intent, and it is for these properties functions of the type `data(MolecularProperty, ChemicalEntity) -> Observation` is most natural.

When the prototype was implemented it was found convenient to include all intensive properties related to chemical substances in the extent of `MolecularProperty`, i.e. also concentration. However, it is evident that a concentration is not a "molecular" property in the same way as a dipole moment. The concentration relates to a specific situation (a sample), while the dipole moment is something valid for all samples of this substance (provided they have the same state).

Furthermore, there is a problem of terminology. The property "volume" could refer to a sample volume (an extensive property not belonging to a particular chemical substance), or to the volume of solvent used in a sample preparation (also extensive, but clearly related to a particular substance).

[240] "Take 1.3 milligram of substance A and dilute it with solvent B to a volume of 100 millilitres."

Fixed Ligand Concentrations

Taken "to the extreme", the approach with observations would suggest that we also store the fixed ligand concentrations of binding experiments (see section 8.7) as `Observation` objects. Then the `fixedLigands` relationship between `Experiment` and `ChemicalEntity` (section 8.7) could be implemented in the database as a *derived* function. However, there is a conceptual difference between a bioactive compound that is binding to the receptor and a Mg^{2+} ion. Hence, the two cases have been kept apart. The function `fixedLigands` will be used for the former, while the latter will be stored as part of an environment (next section).

8.3.3 Environments

An environment consists of a collection of observations. These are such data that are associated with an assay or an experiment, but not stored as attributes to it. For example, the person who performed an assay is stored directly with the `Assay` object while the concentration of an added alkaline earth metal ion would be stored as an `Observation` object under the assay's `Environment`. In some respect the `Environment` is a description of the experimental conditions. However, much of such data (e.g. experimental procedures) should be stored under `Protocol`, which is meant to be a more "standardised" way of describing how the assay was prepared[241]. Most environmental conditions are stored under `Assay`, but a few may suit better under `Experiment`.

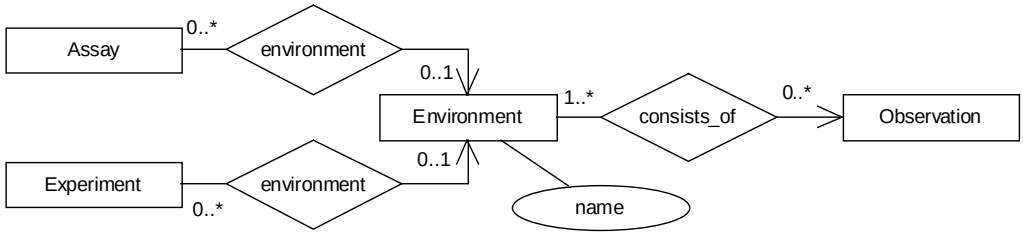


Figure 31. Environments of assays and experiments.

8.4 Binding Assays

Binding assays are a special type of assays where the binding of one or several ligands to one or several binding sites (e.g. cell membrane receptors) are studied. We may perform a variety of different binding experiments with a binding assay, e.g. time binding experiments and concentration binding experiments. (A brief introduction to these aspects of drug design was given in section 2.2.)

We may distinguish between time binding assays and concentration binding assays, but this is not necessary. Instead we do a clear separation of different kind of experiments (see next section). Thus, the kind of information we want to store for an assay is how, when, where, and by whom the assay is performed. In addition to this we store the sample volume, and for a binding assay we store which binding sites we expect the assay to contain, and for which

[241] Presently, the implementation of `Protocol` is the simplest possible: a name and a description, both character strings. Future extensions of this relies on possibilities to interface with lab computers and other electronic equipment. For example, if an XML or CORBA standard schema for chemical, biochemical, or pharmacological laboratory experiments eventually would be agreed upon, the database type `Protocol` could be evolved to follow this standard, and functions for importing such data could be developed.

experiments it was used. Finally, we also store an "environment", which is a collection of observations or measurement data (see section 8.3). This gives us the EER-diagram of Figure 32, with Figure 33 for additional details. (The types `BindingExperiment` and `Environment` will be discussed in other sections.)

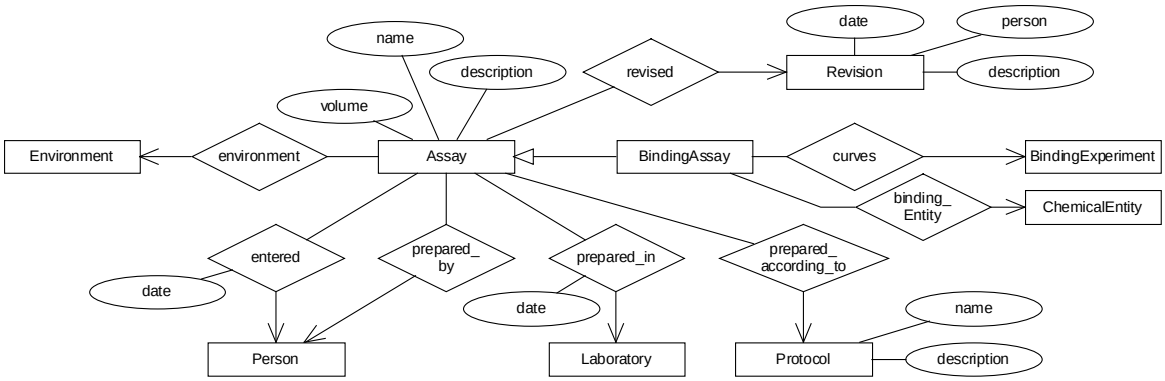


Figure 32. The Assay subdomain.

8.4.1 Revisions

The type `Revision` is intended for information that is obtained after the assay was entered into the database. This could be if, e.g., it was later found that a preparation procedure does not accomplish exactly what was believed at the time of preparation, or if some systematic error has been detected.

8.5 Protocols and References

At the present stage of the prototype, the type `Protocol` is only a text description, and it is used in several places, e.g. for describing the preparation of assays and for describing the set-up of experiments. It is probable that this type could be subtyped, to get better, more specialised, data structures for these descriptions.

Reference objects should hold information on where data has been published, e.g. in a journal article, on the Web, or in a database on CD/DVD. Presently, references are just character strings. There are several alternatives, but probably the modelling solution of some major data source, e.g. MedLine, should be followed.

8.6 Persons and Laboratories

Persons, laboratories, addresses, and countries may be described by Figure 33.

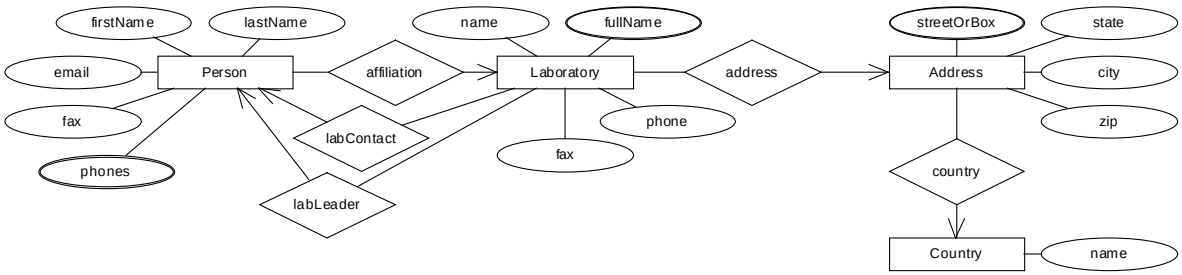


Figure 33. Persons and Laboratories.

8.7 Binding Experiments

As discussed in section 2.2.2, there are different kinds of binding experiments, e.g. time binding experiments and concentration binding experiments. The thing they have in common is that the dependent variable is the concentration of bound radioligand. On the other hand, they have different independent variables: in a time binding experiment the bound ligand concentration is a function of time (a kinetic experiment), while in a concentration binding experiment it instead depends on the concentration of some ligand (a thermodynamic equilibrium has been reached).

With the description of section 2.2 as background, the diagram of Figure 34 may be constructed.

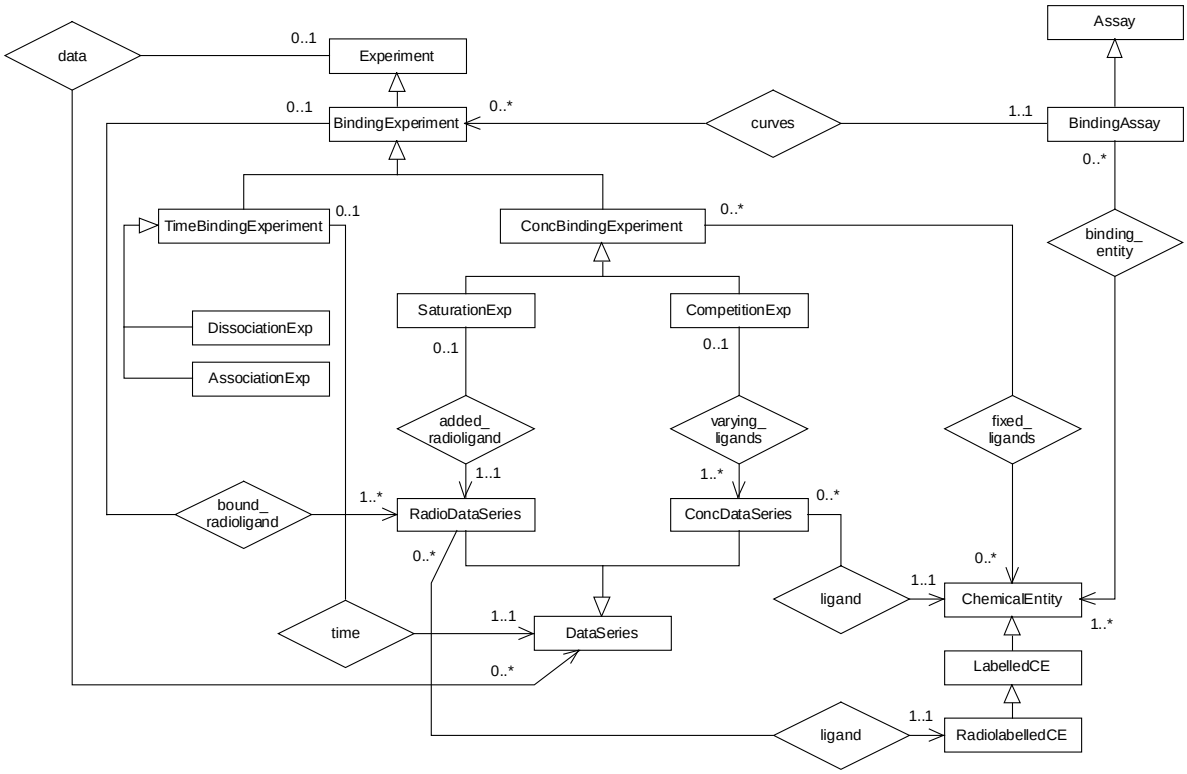


Figure 34. Conceptual schema of binding experiments.

A few constraints that have been introduced here are:

- There may be several "bound_radioligand" `RadioDataSeries` for a `BindingExperiment`. In practice they then refer to parallel experiments, although this is not prescribed by the schema.
- A `CompetitionExperiment` normally has a single "varying_ligand" `ConcDataSeries`, but several are possible.
- There is a single ”added_radioligand” for a `SaturationExperiment`.
- `ConcBindingExperiments` (but not `TimeBindingExperiments`) may have one or several ”fixed_ligands”, additional ligands with fixed concentrations.
- Each `RadioDataSeries` and `ConcDataSeries` refers to a single "ligand" `ChemicalEntity`. Thus, there are not several radiolabelled ligands giving a compound signal.

8.7.1 Implementation of Binding Experiments

When we implement binding experiments in the prototype database we will make a few simplifications:

- We do not distinguish between `RadioDataSeries` and `ConcDataSeries`. Instead we treat a data series that is measured by beta-counters as any concentration data series (as long as we know how to convert from DPM to nM), see section 8.2.4.
- We do not subtype `ConcBindingExperiment`. The two relationships "added_radioligand" and "varying_ligand" are substituted by a single relationship "varying".
- We do not subtype `TimeBindingExperiment`.
- We introduce two types `TBE_enum` and `CBE_enum` to serve as classifiers of time and concentration binding experiments, respectively (vide infra).

Figure 35 shows the stored functions in the implementation of binding experiments. The interfaces of the types `Experiment`, `BindingExperiment`, `TimeBindingExperiment`, and `ConcBindingExperiment` are available as supplementary material. Quite a few derived functions have been implemented in order to help the user to access relevant data, and some of them will be mentioned in this section.

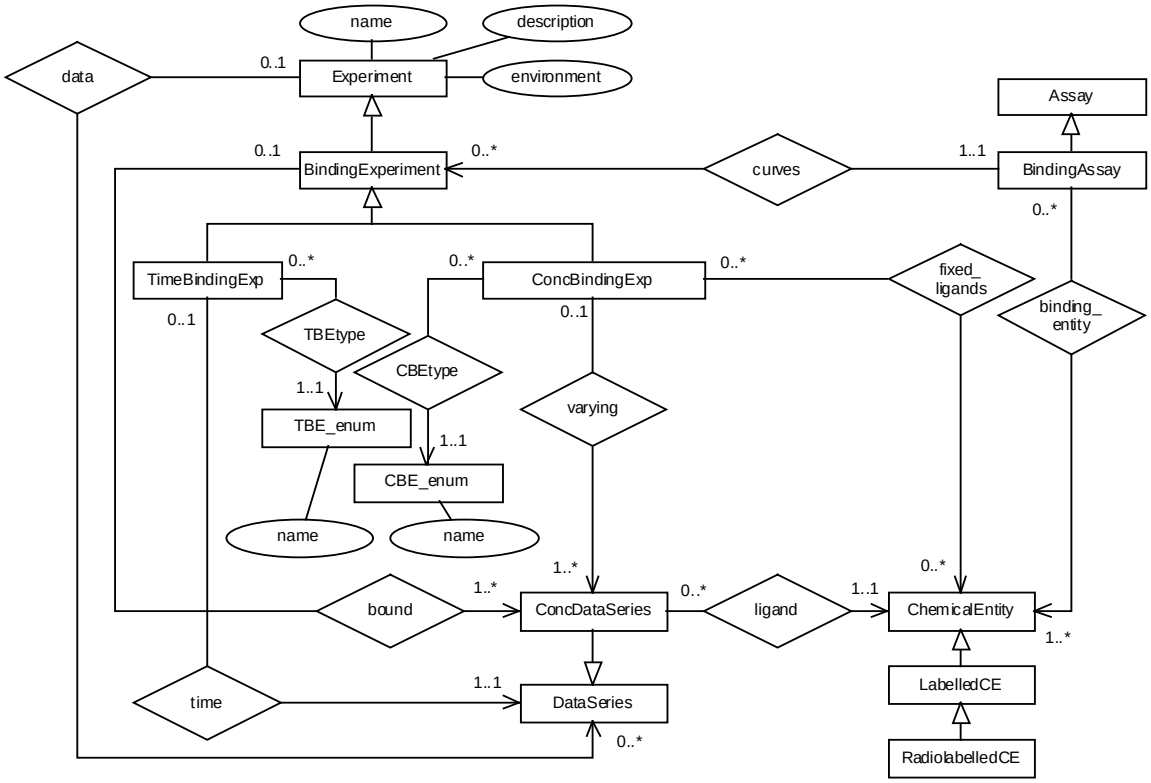


Figure 35. Modified schema for experiments, used for implementation.

”Fixed ligands” are stored as tuples `<ChemicalEntity, Quantity>`, where the quantity of course is the fixed concentration of the chemical entity. To add such a ligand one should use `add_fixed_ligand(ConcBindingExperiment, ChemicalEntity, real concentration, Unit)`, which will check that the ligand is not already stored as a ”fixed ligand” under this experiment[242]. If we want a list of all fixed ligands in an experiment, without their concentrations we may use the function `fixed_ligands(ChemicalEntity)`[243].

To get the average of all `bound` data series for a ligand in an experiment the function `average_bound( BindingExperiment, ChemicalEntity ) -> vector of real` may be used. This function returns the resulting vector in the unit nM.

Finally, a consistency requirement on an experiment is that all its data series should be equally long. This is checked by `consistent(Experiment e) -> boolean`.

Views of All Data Series of an Experiment

For an object `:exp` of type `Experiment` it is trivial to find all its data series, we simply follow the “data” link: `data(:exp);` However, the task is not that trivial if the object’s most specific type is `BindingExperiment` or further down in the hierarchy. Consider for example an object `:tbe1` of type `TimeBindingExperiment`. This object has at least two data series, exactly one data series representing time (`time(:tbe1)`), and one or several data series representing bound radioligand concentrations (`bound(:tbe1)`). Thus, we need to make a union of these, and any possible other data series (which may be stored as `data(:tbe1)`):

[242] I.e., we do not want ligand A to be stored twice, but we *do* allow several different fixed ligands.

[243] This will be useful when we want to find *all* ligands involved in an experiment or assay, irrespective of whether they are fixed or varying. For example, this information is needed when an experiment evaluation is constructed (section 8.8).

```
create function allData(TimeBindingExperiment tbe)
    -> DataSeries ds
    as
    select ds
    where ( ds=data(tbe) or ds=time(tbe) or ds=bound(tbe) );
```

Similarly, we have a function `allData(ConcBindingExperiment)` returning all data series for a concentration binding experiment. To allow for polymorphic calls we also define functions `allData` with arguments of type `BindingExperiment` and `Experiment`.

Views of Independent and Dependent Data Series

PAQS users will probably not work much with the concept of independent and dependent variables. If the need should arise we could split the function `data(Experiment)` in two, or label it with a flag. For the types further down in the hierarchy the situation is clearer, e.g. we know that for objects of type `ConcBindingExperiment` independent (`varying`) and dependent (`bound`) variables are easy to distinguish. However, there is no ”label” explicitly telling an application that `varying` is an independent variable. A situation where the question of independent and dependent variables will be important is when we try to automate plotting. E.g., a reasonable extension of the type `Experiment` would be a function which plots its data series in an xy-diagram[244]. Clearly, we are then a bit particular about getting the independent variable on the x-axis.

Further Classification of Binding Experiments

As we have seen previously in section 2.2.2 there are different kinds of concentration binding experiments, e.g. saturation binding experiments and competition binding experiments. The same applies for time binding experiments, which can be dissociation experiments or association experiments. According to Figure 34 there is no difference between the types `DissociationExperiment` and `AssociationExperiment` and their supertype `Time-BindingExperiment` (except for the classification itself), and hence there is really no need to implement them as three different types.

Instead of subtyping the type `TimeBindingExperiment` as `DissociationExperiment` and `AssociationExperiment` it would be useful to store all time binding experiments as instances of `TimeBindingExperiment`, and to have a label indicating if the experiment was dissociation or association. This is clearly a case where an enumerated type would be useful. However, enumerated types are presently not available in Amos II.

The solution chosen in the prototype is to implement the enumeration as a separate type “TBE_enum”, with one instance for dissociation and one for association experiments. This has the advantage that, in the future, the type can be extended to include some functionality. (For example, each object of this type could store which `FitModel` objects that are appropriate to use for curve fit analysis, if this should differ.) Another advantage of implementing the classification of time binding experiments by a separate type instead of by inheritance is that if a new kind of time binding experiment (with no additional functions) is encountered, it can be stored as a new instance of `TBE_enum`, and the type system needs not be changed[245].

[244] This could be accomplished by shipping the relevant data series to an external function in the same way as when data series are averaged.

[245] To hard-code the experimental types (which are part of the domain logic) into the GUI, would be a poor solution since we would need to change the code and recompile the GUI as soon as we add an experiment type. This also contradicts the principle that domain constraints should be implemented in the database schema, and not in application programs.

In the case of the two types of concentration binding experiments, we see that the types `SaturationExperiment` and `CompetitionExperiment` in Figure 34 differ somewhat from each other. However, since we presently choose not to distinguish between `RadioDataSeries` and `ConcDataSeries` we may without problems use the schema in Figure 35, with `CBE_enum`.

8.8 Experiment Evaluations

When we take a competition binding experiment and analyse it with a curve fitting program to get the affinity constants for all ligand/receptor pairs, we ”evaluate” the experiment. This section discusses how such experiment evaluations can be modelled, and how they actually could be performed *by* the database system.

In section 8.8.3 we temporarily depart from object-orientation. An alternative implementation of the ”Strategy” design pattern will be discussed. This alternative approach is not strictly object-oriented, but relies on storing the name of a function as an attribute to a type.

First of all we may note that the usual approach to perform an experiment evaluation is to take *one* experiment at a time as input to a curve fitting program. Wikberg's fit program BindAid can take several experiments in a ”multi-curve fit”, and the prototype database will be able to represent this more general case.

Secondly, there are different kind of fit parameters. If we analyse a concentration binding experiment the main purpose is to get values for the affinity constants. However, usually a number of other parameters are determined together with the affinity constant. The prototype will store all these parameters, not only the fitted affinity constants. Different approaches to representing these different kind of fit parameters are discussed in section 8.8.6.

Thirdly, we should consider how many of the tasks of a fit program the DBMS shall be able to take over. In section 8.8.1 we will see how the implemented database schema can construct the an appropriate set of fit parameters for a chosen "fit model", and in section 8.8.2 we will see how the correct weights can be calculated. Thus, it would be possible to write the fit algorithms as foreign functions (in Java or C) and start the fits from within the database. (Even better would be to make the foreign function an interface to an *existing* fit program, and not recode the algorithms.) The prototype does, however, not perform the actual fit.

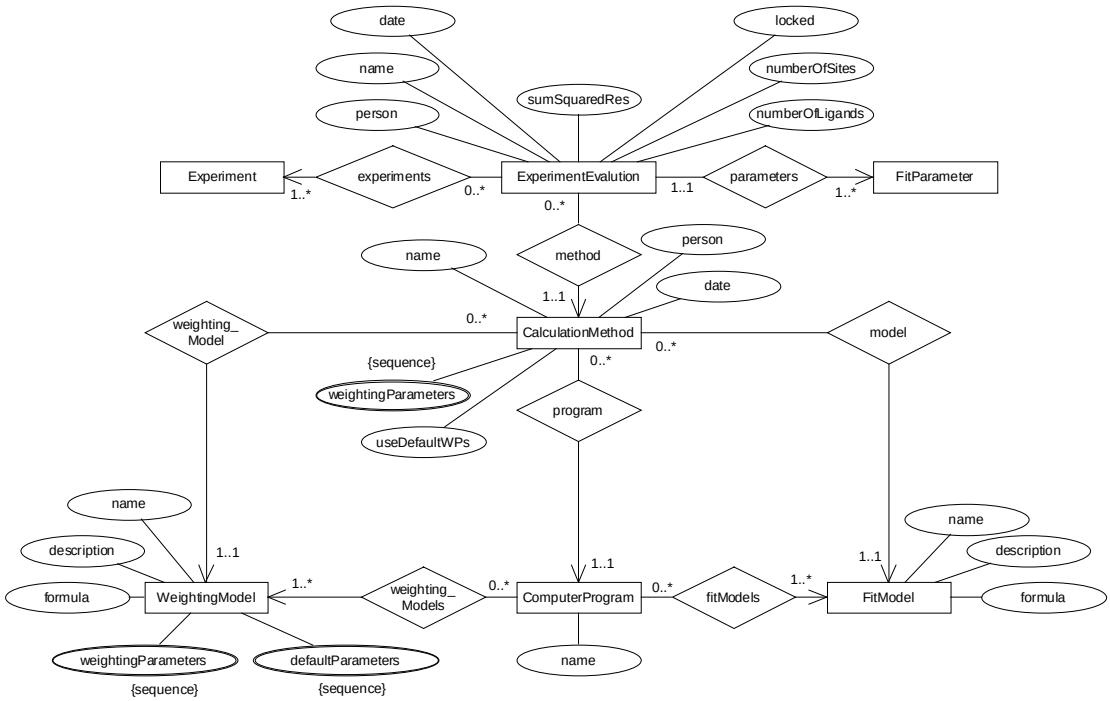


Figure 36. Experiment evaluations connect fit parameters with experiments and calculation methods.

Figure 36 summarises the introduction above and will serve as starting point for the following sections. Objects of type `ExperimentEvaluation` thus represent "runs" or executions of a curve-fitting program[246]. On the other hand, the details about program, fit model, weighting scheme et cetera are handled by the type `CalculationMethod` (8.8.4).

8.8.1 Fit Models

A fit program implements one or several fit models, where a fit model is some equation with parameters. These parameters are adjusted in the fit in order to make the equation represent the experimental data as accurately as possible. Section 2.2.2 gave the equations for a few such fit models (as implemented in the fit program BindAid). However, the number of fit models used in different laboratories are large. It is not realistic to try to define these models once and for all, and it must be possible for a user to submit data to the database without specifying a fit model.

Note that in the approach chosen here, objects of the type `FitModel` represent algorithms for performing fits, not the results of such fits. Thus, the results of an experiment evaluation are *not* kept as attribute values in a `FitModel` object, but as separate `FitParameter` objects.

Most fit models will have a varying number of associated parameters to fit depending on the number of binding sites and ligands[247]. For example, assume we are analysing a competition binding experiment which involves two ligands L_1 and L_2 and an assay containing one binding site B. We decide to analyse this experiment by the fit model called ”Bindfit” (see

[246] Other types of experiment evaluations are also possible, e.g. the visual inspection of a curve, but we will not discuss these here.

[247] The exceptions are models which simply try to fit the experimental data to a predefined curve shape, e.g. exponential, logarithmic, or sigmoid. Such models, too, can be used in the prototype.

section 2.2.2). There are then five fit parameters: the concentration of B, non-specific binding constant for L_1 and L_2, and the affinity constants for L_1/B and L_2/B.

The prototype will be able to combine a set of experiments with a fit model, and then construct the correct number of fit parameters. However, the prototype will not actually perform the fit. Instead the user may enter the fit results manually. It would be straightforward to extend the prototype to perform the fit too, as Amos II foreign functions. However, this will not be an interesting feature until there are good interfaces both for import of experimental data and for user interaction (graphically).

Creating the Correct FitParameter Objects

The behaviour which differs between `FitModel` objects in the prototype is the way fit parameters are created. The function `setupFitParameters(ExperimentEvaluation)` will return a bag with the correct `FitParameter` objects for the fit model chosen. This is done in the following way:

- The `ExperimentEvaluation` object (`:ee`) first collects ligands and binding sites in two bags. (This information is of course taken from the appropriate `Experiment` objects.) Then `:ee` calls upon its `CalculationMethod` (`:cm`) to create fit parameters: `createFitParameters(CalulationMethod, bag ligands, bag sites)`.
- `:cm`, in turn, knows which fit model that shall be used, and calls upon the appropriate object `:fm` of some subtype to `FitModel`: `createFitParameters(FitModel, bag ligands, bag sites)`.
- Finally, `:fm` knows which fit parameters are needed, and has the appropriate behaviour implemented in `createFitParameters`, either as a database procedure or as a foreign function.

8.8.2 Weighting Models

Many fit programs allow a user to assign different ”weights” to data points. If the fit program works by minimising the sum of squared residuals between experimental and calculated values, then each such residual may be given a weight to indicate its importance in the target function. The prototype knows how to calculate these weights for a small number of weighting schemes. Since many such schemes take the experimental data value into account the algorithms for generating weights take a vector of real values as input. The output is a vector consisting of the weights, *not* the weighted values[248]. (It is usually the residuals that are weighted, not the experimental data.)

The type `WeightingModel` has the attributes name, description and formula, exactly as `FitModel`. Each `WeightingModel` object has two sequences: `weightingParameters` and `defaultParameters`. These are simply the *names* of the parameters and their default values. The meaning of the parameters may be inferred from the attributes `description` and `formula` of the `WeightingModel` object. A `CalculationMethod` object may also have a sequence `weightingParameters`, these are the actual values used in the calculation. If the `WeightingModel` does not contain any parameters[249] there are no `weightingParameters` on

[248] The function `multiply_vectors(vector v1,vector v2) -> vector of real` will multiply any two vectors of equal lengths, element by element.

[249] For example, no weighting parameters are needed when all weights are unity, or equal to the inverses of the data values.

the `CalculationModel` object, and the same is true when the default weighting parameters are used. In the latter case the boolean flag `useDefaultWPs` is true[250].

Three weighting schemes were implemented in the prototype, all taken from the fit program BindAid (Wikberg 2001). Note that the weights should be multiplied to the respective residuals in the target function. The three weighting models were implemented as subtypes of the type `WeightingModel`, with different implementations of the function `getWeights`. In section 8.8.3 and Appendix E an alternative implementation is discussed.

No Weights: Type NoWeighting

All weights are unity. The two vectors holding names and default values of weighting parameters are empty. The function `getWeights(NoWeighting, vector y, vector wp)` returns a vector of equal length as the vector `y`, and with all elements set to 1.0.

Inverse Squares: Type MunsonRodbardWeighting

The variance of the measured data value is estimated to be equal to the inverse of the square of the data value (Munson and Rodbard 1980). These variance estimates are used as weights. The two vectors holding names and default values of weighting parameters are empty. The function `getWeights(MunsonRodbardWeighting,vector y,vector wp)` returns a vector with elements $w_i = 1/y_i^2$. (The vector `wp` is not used.)

A Three-Parameter Model: Type DeLeanWeighting

A weighting model with three parameters introduced to avoid unreasonably high weighting factors for data values close to zero (De Lean, Hancock and Lefkowitz 1982). The vectors holding names and default values are {"a", "b", "c"} and {0.000001, 0.001, 1.5}, respectively. The function `getWeights(DeLeanWeighting,vector y,vector wp)` returns a vector with elements $w_i = 1/(a + b \times y_i^c)$, where {a,b,c} are the values of the vector `wp`.

8.8.3 Two Approaches for Implementing the Strategy Pattern in Amos II

A `FitModel` object has state (the values of its attributes `name`, `description`, and `formula`) and behaviour (methods to create the correct type and number of `FitParameter` objects). The behaviour is different for the different `FitModel` subtypes, and we may consider the types `ExperimentEvaluation`, `CalculationMethod`, and `FitModel` to constitute a "Strategy" design pattern[251], see Appendix E. The same applies for `WeightingModel`, where the behaviour which differs between subtypes is the way a vector of weights is calculated.

The implementation described above and in Appendix E does work, but the number of subtypes will per definition grow linearly with the number of fit models. Thus, there will eventually be quite many subtypes of `FitModel`, and this will become unsurveyable. Another major disadvantage with the approach chosen above is that a user who wishes to add a new fit model needs to add a new subtype to `FitModel`, i.e. change the database schema.

[250] Since users will be allowed to manipulate the database by AMOSQL commands it could by accident happen that the `useDefaultWPs` flag set true at the same time as `weightParameters` contains real values. However, the method `getWeightingParameters(CalculationMethod)` will fetch the default parameters as soon as `useDefaultWPs` is set true.

[251] Gamma, Helm, Johnson, Vlissides 1995; Grand 1998.

The weighting models are implemented in the same way, and suffer from the same problems, although it is not expected that the total number of weighting models will be as large as the number of fit models.

An alternative approach would be to store each fit model as an instance of type `FitModel`, and each weighting model as an instance of `WeightingModel`. Thus, no subtypes should be used. This approach is straight-forward if the types should only have state, but no behaviour. But we want to let each `FitModel` object know which `FitParameter` objects to create, and we also want to let each `WeightingModel` object know how to calculate the weights for a given vector of real numbers. Eventually, we might want to let `FitModel` objects be able to perform the fit too, probably by means of Amos II foreign functions.

Thus, we want to benefit from the object-oriented approach and include operations, but we don't want to use inheritance. One way to accomplish this is to use a flag to classify the `WeightingModel` objects as one of the three kinds mentioned above. The type `WeightingModel` could then implement `getWeights` by means of `if/then/else`-controlled conditional execution[252]. This is not a very beautiful solution. A more serious disadvantage is that a user who wants to add a weighting model needs to change *existing* code, adding another `else/if/then` clause. (With the inheritance solution the user adds another subtype, but does not change existing AMOSQL code.) Hence, this procedural approach is rather poor.

A third solution is now possible in Amos II[253]. Amos II functions are first-class objects in the type system, instances of the meta-type `function`. With the Amos II function `function-named( charstring ) -> function` it is easy to get hold of a function from its name. Thus, first we implement a method to calculate the weights for each weighting model, but in contrary to the OO polymorphism/inheritance solution we give each implemented function a name of its own. Secondly we store the *names* of the implemented functions as attributes `getWeightsFunctionName` of the corresponding `WeightingModel` objects. Now, all we need is a means to *apply* a function if we have a handle to it[254]. This is accomplished by the foreign function `apply( function f, vector parameters) -> vector results`, which was implemented in Lisp by Tore Risch for this project. See Appendix F for a detailed example of how to use the function `apply`, and a comparison with the Java method `java.lang.reflect.Method.invoke(Object, Object[])`.

With this new approach, the functions actually calculating the weights are decoupled from the `WeightingModel` type[255]. A function for calculating weights takes two vectors as arguments (values and weighting parameters) and returns another vector (weights). The function thus has no connection whatsoever to the type `WeightingModel`. However, any sensible name given to such a function should allude to weighting in general as well as the specific model. E.g., a function name may be ”getDeLeanWeights”.

Now, what a user needs to do in order to store a new weighting model (or a new fit model) is to create an instance of `WeightingModel` (`FitModel`), including a name for the function to do

[252] In many programming languages a switch/case construct would be used instead of if/then/else.

[253] The approach described here does presently not work with the official Amos II version available at the Amos II web site. The "developers" Amos II version is needed, where a user can go out to Lisp.

[254] Usually, we apply (or invoke) a function by writing its name in an AMOSQL statement, i.e. `name(:donald);` applies the function `name` on an object `:donald` (presumably of type `Person`). In an OO system a method invocation might look like `donald.name();`. However, in the functional language Lisp higher-order functions such as `(apply function arg_list)` are often quite useful (Haraldsson 1993).

[255] In an OO system object behaviour is implemented as methods of a class, and in the Amos II functional model as functions taking the object's type as an argument.

the calculations, and then implement this function. The implementation can be as a foreign function, in Java, whence the function implementation in Amos II could be very brief:

```
create function getDeLeanWeights( vector values,
    vector weightingParameters ) -> vector of real
    as foreign "JAVA:Prot1Weighting/deleanWeights";
```

This last ”apply function” approach has the advantages that

- there will not be a plethora of subtypes to `FitModel`,
- a user needs not introduce new types, and
- the user needs not change any existing AMOSQL code.

One disadvantage may be that the approach is less object-oriented than the normal Amos II data model. However, we have accomplished the same functionality as the OO polymorphic solution with subtypes, but without using the procedural `switch/case` or `if/then/else` constructs. There ought to be no significant loss of performance since the extra level of indirection should be considerably less time-consuming than the algorithms used in the "concrete strategies".

8.8.4 Calculation Methods

A calculation method represents the way a calculation was performed, not the actual result. For a curve fit this implies that by looking up a calculation method we should be able to determine which model (section 8.8.1) that was used to fit the experimental data, which weighting scheme that was used (section 8.8.2), and the setting of any parameters that determine the execution of the fit program. We do not expect to find the values of the fitted parameters here, they should instead be related directly to an `ExperimentEvaluation` object.

In section 8.8.2 we saw that a weighting model may use a few parameters to calculate the weights. For each such ”weighting parameter” there is presumably a default value, stored under the corresponding `WeightingModel` object. Under `CalculationMethod` we store a boolean `useDefaultWPs` , and, if `useDefaultWPs` is not `true`, a vector with the weighting parameters actually used.

If we apprehend a calculation method as a combination of a fit model, a weighting model, and a set of weighting parameter values, then we will often want to use the same combination over and over again. For a given fit program, it is likely that only a few such combinations are ever used. Therefore it is useful to store each `CalculationMethod` object with a special name, and also to know who created it and when.

8.8.5 Computer Programs

The schema in Figure 36 has each `FitModel` associated to several `ComputerPrograms`, and vice versa. Clearly a single program may implement several fit models. What is more difficult to solve, however, is whether a model can be implemented by several programs. Take two programs from two different origins, e.g. BindAid (Wikberg 2001) and SigmaPlot (SigmaPlot 2001). These two programs may very well implement the same model, under the same name or under different names, but in slightly different ways. Here, we take a practical approach: If the difference is such that one program has an additional fit parameter, then we say that the two programs implement *different* models (even though they may claim to implement the same model). Furthermore, two implementations may differ in the way they solve the

numerical optimisation problem. Then, the fit model is still the same, but the final result could be quite different[256]. We still consider this as one and the same `FitModel` object. That the implementations may differ is handled by the relationship between `CalculationMethod` and `ComputerProgram`[257]. Thus, we could say that it is the `formula` attribute of `FitModel` that determines it[258].

The type `ComputerProgram` should include attributes on the program's origin, i.e. by whom it was written (person + laboratory, or company). Program versions could also be handled explicitly instead of having each version as a separate program (the present solution).

8.8.6 Fit Parameters

Assume that we have an assay which contains one binding site B, and that the binding experiment we are analysing involves two ligands L_1 and L_2. The `Bindfit` fit model then has five fit parameters: the concentration of B, non-specific binding constant for L_1 and L_2, and the affinity constants for L_1/B and L_2/B. The schema to be developed in this section will represent all these parameters, not only the fitted affinity constants. We have already in Figure 36 seen that the fit parameters will be represented as `FitParameter` objects associated with an `ExperimentEvaluation` object. In this section we will look closer at the `FitParameter` type.

Different Kind of Fit Parameters

Conceptually, the fit parameters in a program such as BindAid differ from another in two ways (see Figure 37): What property they refer to (i.e. what, in the real world, we try to fit a value to), and under which constraints they are used in the fit program. The constraints occurring in BindAid are "none" (the parameter is free to float in the fit), constrained to be equal to another parameter, and held constant[259]. If we use the `Bindfit` model of BindAid there are three kind of parameters: one concentration for each binding site (or receptor), one non-specific binding parameter for each ligand, and one affinity constant for each ligand/binding site combination.

Thus, in Figure 37 we would like each `FitParameter` to be one of the three subtypes at the bottom, and also to be one of the three subtypes to the right. The two hierarchies are orthogonal to each other, and to implement this multiple inheritance situation as separate types in Amos II would require 9 additional subtypes[260]. This is clearly not the way to proceed, especially when we consider that there are many more alternatives: In the hierarchy to the right we should include other properties for other types of data analysis, e.g.

[256] For example, the two programs could very well find two different local minima even when they start from the same point of the optimisation problem hypersurface.

[257] In the implemented prototype no check is made that all `ComputerProgram` – `FitModel` links are present. I.e., a `CalculationModel` object `:cm` may very well be linked to a `FitModel` object `:fm` and a `ComputerProgram` object `:cp` without `:fm` and `:cp` being linked. This will not be a problem. The main benefit of the `ComputerProgram - FitModel` relationship will be when the database has been populated with known programs and models, and it is accessed through a graphical user interface.

[258] In principle, `formula` could be declared a key in the Amos II implementation. However, it will occasionally be practical to be able to use `FitModel` objects without formulas, and therefore I have chosen not to make formulae unique.

[259] Further possibilities for fit constraints exist: proportional to another parameter, the average of several other parameters, and so on.

[260] Amos II allows for multiple inheritance, but an object is required to have a most specific type (3.2.6).

dissociation rate constant and initial bound concentration for `DissocFit` calculations. In the lower hierarchy other types of fit constraints (e.g. averaging) are possible.

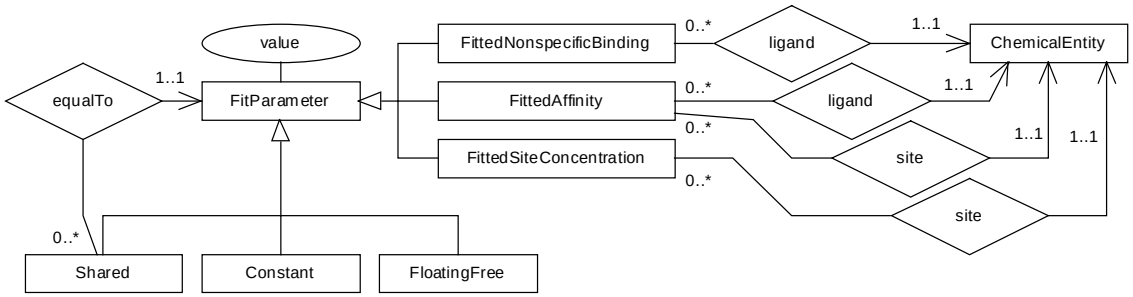


Figure 37. Conceptual schema of different kind of fit parameters (for the Bindfit model of BindAid).

The further modelling and implementation of fit constraints is described in section 8.8.7.

In the ("property") hierarchy to the right of Figure 37 the only difference between the subtypes is whether they are linked to ligands, receptors, or both[261]. If we are willing to give up the control of this linking we can easily exchange the hierarchy to the right of Figure 37 for something simpler, see Figure 38.

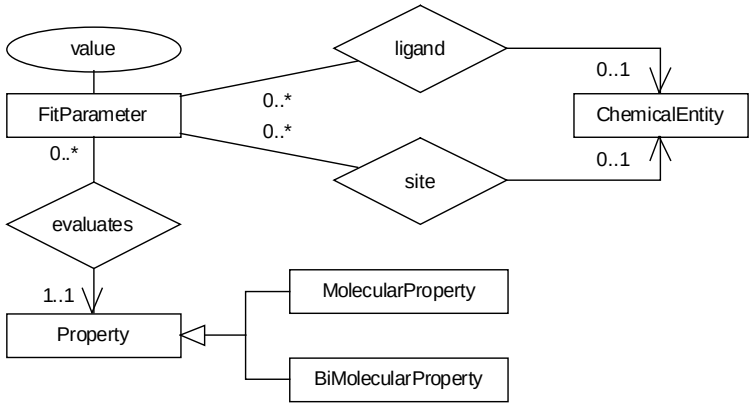


Figure 38. How the prototype relates fitted properties and chemical entities to fit parameter.

We subtype `Property` to `MolecularProperty` and `BiMolecularProperty` (see section 8.3), but never use this information explicitly to control the linking of `FitParameter` objects. Instead we rely on the users and the applications to use "factory methods" for creating the correct `FitParameters`. An example of this is described in connection to the `FitModel` type in section 8.8.1. It would also be possible to use triggers in the database to make sure that only correctly linked `FitParameters` occur. Thus, we have arrived at a situation where we have subtyped `Property` instead of `FitParameter`.

Limitations of the Suggested Schema

The suggested schema in Figure 38 has the constraint that a `FitParameter` may be linked to at most one receptor and one ligand. This is adequate for binding affinity constants (R + L ⇌ RL), dissociation rate constants (RL → R + L), and, as far as I have been able to see, all other

[261] Some fit parameters should have neither receptor nor ligand links, e.g. the P-parameter (or pseudo Hill coefficient) of the logfit model.

fit parameters encountered in the BindAid program. However, the schema is obviously not suited for descriptions of general chemical kinetics or equilibria.

As pointed out above, the schema does not catch the full semantics. I.e., we know that a fitted binding affinity is valid for exactly one ligand and one binding site, but this is not proscribed by the schema. This is something I find unfortunate, but one advantage is that we do not need to create new subtypes of `FitParameter` or any other type when a new kind of fit parameter is introduced. All we need to do is to create a new object of type `Property` (or one of its subtypes).

An alternative solution would be to use three generic subtypes to the type `FitParameter`, one for the kind of fit parameters which only should be linked to a ligand, and so on (see top of Figure 39). This would allow new kind of fit parameters to be added without schema changes. A second alternative (bottom of Figure 39) would be to have a single function mapping from `FitParameter` to `ChemicalEntity`, but with an additional attribute indicating the "role" which the chemical entity plays. (This would be similar to an association class in UML). The first of these alternatives seems quite attractive.

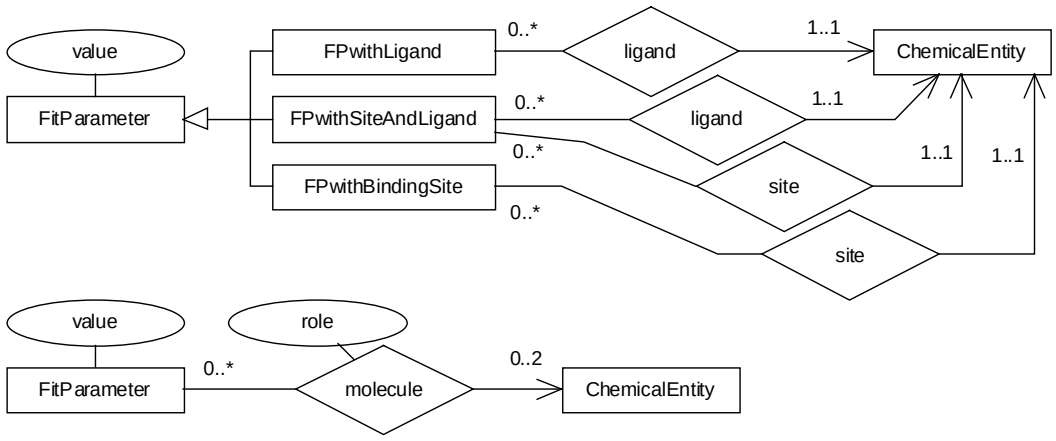


Figure 39. Two alternative schemas for fit parameter, keeping some more of the domain semantics.

A further potential disadvantage with the solution presented here is that there could be a large amount of small objects. However, the `FitParameter` objects should be stored in external data sources, not in the mediator. Thus, even if all experiment evaluations were performed from the mediator, the `FitParameter` objects would be stored externally (e.g. on XML files) and deleted from the Amos II system once a satisfactory fit was accomplished. When need arises for a parameter, e.g. a binding afffinity it can be accessed from the external source.

The Vector Alternative

Several alternative designs where vectors are used have also been investigated. For example, an `ExperimentEvaluation` object could have five vector attributes: `bindingSites`, `ligands`, `evaluatedProperties`, `parameterValues`, and `fitConstraints`. Then the vector indices would couple ligands and receptors with the correct fit parameter values through some algorithm[262]. However, such schemes will be quite complex and inflexible. Furthermore, the OO approach of Figure 38 will suit better with import and export of XML data. (An advantage with most vector approaches is that the user-defined types get more cohesive.)

[262] Suppose we have *N* ligands and *M* binding sites. Then, if the "Bindfit" model is used, there will be *N*M* binding affinities, N non-specific bindings, and M site concentrations in each of the latter three vectors. However, if another fit model is used there will be a different number of fit parameters.

8.8.7 Fit Constraints

There are some important differences between fit parameters which are subject to different fit constraints (free, constant, shared,, averaged,…). For free-floating parameters we would like to store the initial value and an estimated error, and for shared parameters a link to another fit parameter. Constants would need neither of these. (On the other hand, an "average of" constraint would imply the need to refer to a *set* of other parameters.) For all constrained parameters it should be possible to store a textual motivation on why the constraint was introduced, or from where the value was taken.

First of all, we may discriminate between those fit parameters that are free to float and those that are constrained in some way. This can be modelled conceptually as in Figure 40:

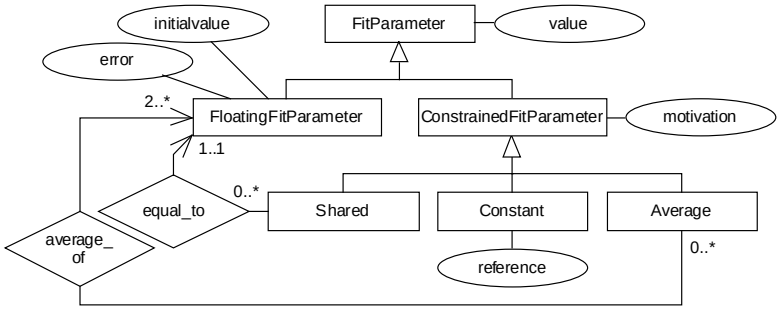


Figure 40. A schema which differentiates parameters free to float from others.

Alternatively, we can do as in Figure 41 *and* in the prototype: We get rid of the constraint hierarchy of `FitParameter` by delegating it to a separate type `FitConstraint`.

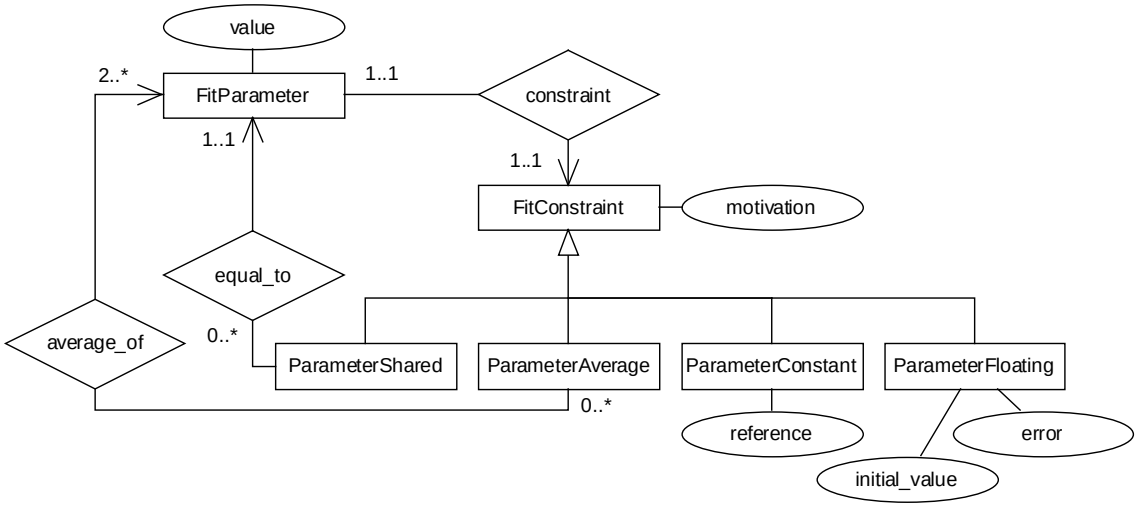


Figure 41. Constraints on fit parameters handled by delegation. This is the approach chosen in the prototype.

This solution will make it easy to let the schema evolve to include new fit constraints.

9 Remaining Issues for the Implemented Prototype

In this section 9.1 the parts of the problem domain that remain to be modelled will be discussed. Then, section 9.2 will describe some work of a more technical or programming art which needs to be done in order to make a more interesting prototype of PAQS. In the last sections a few suggestions to improvements of the Amos II mediator system will be given.

9.1 Remaining Subdomains

As mentioned above (section7.2), some subdomains are only represented by stubs, and some have not been modelled at all. In the following subsections a few comments on these subdomains are given.

9.1.1 References

References can be of many types, e.g. they could refer to printed literature (books, journals), web sites, compilations in databases, or private communications. In the presented schema all references are represented by a single type `Reference`, which only has a single attribute `description` (a character string). Clearly, the type should have much more structure, and probably the way to proceed is to model references in the same way as some major data source or integration effort. Since PubMed (5.5.1) is an important resource when it comes to publications, I believe this would be the choice. Then, integration with BIND (5.4.1) would also be easy.

9.1.2 Protocols

Protocols are descriptions of some experimental (or calculational) procedure. For example, a protocol could describe how a sample was prepared. Usually, several (many) samples would be prepared according to the same protocol. The presented schemas have a type `Protocol` with two attributes, a `name` and a `description` (character string). The type will have to be modelled in more detail.

9.1.3 Chemical Entities

`ChemicalEntity` is the type used to denote all small organic molecules, large biopolymers (proteins, DNAs), and inorganic salts. I.e., it is used for ligands, binding sites, and possible additives in the assay. The type representing these very different kind of chemical species presently only has the attribute `name`. Although `ChemicalEntity` is subtyped as `Labeled-ChemicalEntity`, which in turn is subtyped as `RadioLabeledChemicalEntity` these subtypes have no additional attributes.

Obviously, there should be many more attributes, e.g. representing label position, molecular structure, and physico-chemical properties. For proteins the amino acid sequences must be represented, and there should be effective means for searching the underlying data sources for a particular amino acid sequence, and for a chemical substructure.

The idea behind using a subtyping hierarchy is the following: Suppose we wish to label a hormone (called *HormH*) by exchanging a specific hydrogen atom (*H*) for the radioactive

isotope iodine-125 (^{125}I). We could then store the hormone (*HormH*) as a `ChemicalEntity`, the hormone with iodine as a `LabeledChemicalEntity` (*HormI*), and the radioactive substance we use in the binding experiment as a `RadioLabeledChemicalEntity` ($Horm^{125}I$). Thus, the position and the atomic number of the label should be attributes of `Labeled-ChemicalEntity`, while the radioactive properties should be stored under `RadioLabeled-ChemicalEntity`. In this way, each labelled substance can refer back to a "normal" (unlabelled) substance. Thus, although *HormI* is itself a "normal" substance (with a IUPAC name, a CAS number et cetera), it is easy to store that it "originated" from *HormH*.

Obviously, other design solutions are possible, too, e.g. with a radiolabelled substance being a composition of a chemical entity and a label. However, I believe it is important to keep the possibility for polymorphism.

9.1.4 Descriptors

The one subdomain of Figure 17 which has neither been included in the modelling of chapter 8 nor implemented is the one called "Descriptors".

In a proteo-chemometric multivariate analysis each `ChemicalEntity` object must be described by some descriptors. These descriptors will be different for ligands (small organic molecules) and binding sites (amino acid sequences). They will also be different for different investigations, and within one and the same investigation some testing of new descriptors will often be made. Thus, we need a general and flexible way to organise descriptors in the database.

I have not presented a database schema for descriptors in this Thesis, but I believe the following two points are important to consider: (1) Descriptors may be stored in the database as numbers or as bitstrings. (2) There are two alternative kinds of descriptors, depending on their origin. To elaborate on point (2):

- Some descriptors are *generated* as part of the investigation. An example is the set of 24 binary descriptors used for 4-piperidyl oxazole derivatives[263]. Each of these only has a meaning as a member of the set, together with the definition. Therefore it seems appropriate to view a coherent set (e.g. called "DescriptorSet") as the central type here, with the descriptors stored as attributes.
- On the other hand, physico-chemical descriptors are more *self-supported*. Take for example the lipophilicity, the dipole moment, and the van der Waals surface area of a ligand molecule. These are experimentally or theoretically determined properties, and they have a meaning without being included in a set of descriptors. However, the values stored (e.g. in external databases) most certainly have to be transformed or normalised in order to be used in a multi-variate analysis, and hence it is fair to say that such descriptors too will be used in coherent descriptor sets.

9.2 Technical topics

There are a range of more "technical" topics which need to be considered to make PAQS work as intended. Wrappers (9.2.1) will be needed to get access to external data sources and indexes (9.2.2) will help to get a good performance.

[263] Lapinsh, Prusis, Gutcaits, Lundstedt and Wikberg 2001.

9.2.1 Wrapping Data Sources

An important aspect of the mediator-wrapper architecture of PAQS and similar systems is the use of wrappers for importing external data (see sections 4.4.2 and 7.1.2). Such wrappers have not been implemented in the Thesis work, but a number of potential data sources have been identified (chapter 5).

From the development described in chapters 5 and 6 I find it likely that three wrapper types will be needed for PAQS in the near future: (i) A general wrapper to XML data, which can be customised for specific markup languages, e.g. BSML. (ii) A wrapper for web forms. (iii) A wrapper for relational databases[264]. All three of these do already exist, although it is only the ODBC/relational wrapper which is currently used "in production".

9.2.2 Indexing and Clustering

The Amos II system presently incorporates two aspect of physical database design: indexes and clustering. As mentioned in section 3.1.2 indexing is important for helping the DBMS find the correct object fast. In traditional, disk-based, database systems, clustering is another means to speed up data access by storing related data together. With Amos II, a main-memory DBMS, clustering is instead used to decrease the database size.

Indexing will certainly be an important topic for the PAQS system, especially if extended to the contents of external sources. In this Thesis, indexes have only been used to define cardinality constraints. No additional indexes for performance have been added.

In the present Amos II system hash and B-tree indexes are implemented. It is likely that the full-scale production PAQS will require some additional index type(s), particularly for retrieval on protein and chemical substructures.

Clustering will be important only for those functions and types where large volumes of data are stored locally in the PAQS system. Most of the data will, however, be stored in external data sources, and for these clustering is not interesting.

9.2.3 Visualisation

Visualisation is a great help in any data analysis. First of all, it would be quite helpful with a simple plot routine for data series (8.2). This would also be rather easy to implement as a foreign function in Java. A more ambitious project would be to provide for advanced visualisation of data mining routines, i.e. the proteo-chemometric multivariate analysis. Here, an alternative to an implementation from scratch would be to wrap an existing visualisation package by a Java class.

A very useful feature would be if a user could sketch a molecular structure on a canvas and then search for the structure in PAQS (see the Binding Database, section 5.4.4).

[264] Although no web data source described in chapter 5 actually allows direct access via ODBC, this wrapper will be needed for databases installed locally, e.g. for purchased databases of chemical substances.

9.3 Suggestions for Amos II

To conclude this section of remaining topics a "wish list" of features for the Amos II system is presented. These features are not necessary for a successful implementation of PAQS, but they would simplify the development of the system and/or render the final system more effective.

9.3.1 Security Matters

Authorisation

The usability of PAQS would be greater if users were allowed to put complex AMOSQL questions to the system, and not only use fixed-form queries via a web page. However, if a command-line interface is provided on the web, one must first make sure users cannot delete local data of other users, or change the database schema. Most commercial DBMSs have authorisation mechanisms for this, but no such mechanism yet exists for the Amos II system.

Encapsulation

Encapsulation is implemented in OO programming languages (e.g. Java and C++) as visibility modifiers. For example, if a Java method is declared `private`, it can only be used from within its class. Similar primitives in Amos II would be useful if users have direct access to the system via AMOSQL. For example, in section 3.2.6 it was described how enforcing more complicated cardinality constraints is a problem in Amos II. If the regular `create` statement could be made non-accessible for users they would be forced to use special "constructor" functions.

Future revisions of the SQL:1999 standard will probably contain different levels of encapsulation[265].

9.3.2 Vector Operations

The Amos II data type `vector` makes it possible to store objects (surrogates or literals) in an ordered sequence. This is particularly useful for scientific and technical databases. However, there are presently quite few operations available for vectors (creation, access of an element by index number, concatenation of two vectors). Special-purpose vector manipulations can be implemented as foreign functions (see Appendix C), but it would be reasonable to include a larger number of general-purpose vector functions in the Amos II distribution, e.g. substitution of an element.

It is possible to put an index on a vector as a whole, but not on individual vector elements. This does presently not seem to present a problem for PAQS.

9.3.3 Abstract Types

Two object-oriented features missing in the Amos II functional OO data model are encapsulation (9.3.1) and abstract types.

An abstract class in Java is a class of which one cannot create objects (the class is non-instantiable). In short, abstract methods of the abstract class are implemented in subclasses,

[265] Eisenberg and Melton 1999a.

and these subclasses are then "concrete" (i.e. instantiable, not abstract). This is a useful concept both in modelling and implementation of an OO system. A corresponding construct in Amos II would be to have abstract types, and abstract functions.

The concept of abstract types has found its way into the new SQL:1999 standard[265], where types may be specified as "instantiable" or "not instantiable".

9.3.4 Miscellaneous

Comments on One Line

Text enclosed by /* … */ is treated as a comment in AMOSQL scripts. However, such comments cannot be nested, and thus it is difficult to temporarily comment out a large portion of a script where a small part (e.g. one line) is already commented out. A nice feature would be to allow a special one-line comment syntax, too.

Clearer IUT Interface and Documentation

It is far from straight-forward to work with integration union types (IUTs). The implemented syntax is not the same as in the examples of the relevant publication (Josifovski and Risch 1999), and it seems that not all combinations of types are possible. A better documentation of this feature is needed, particularly since this information integration aspect of Amos II is very important.

Sorting

Standard SQL allows sorting and grouping of query results. At least a sort function would be useful in AMOSQL, too. This probably applies to most problem domains, not just PAQS.

10 Conclusions

Integration of life science data sources is a very active research area. Some important trends which have been described in chapters 5 and 6 are the use of XML and CORBA as media for information exchange and the use of mediating middleware for integration.

When XML and CORBA are compared, it is obvious that XML has the larger momentum, especially the Bioinformatics Sequences Markup Language (BSML, 5.2.4), supported by the Interoperable Informatics Infrastructure Consortium (I3C, 5.3.4). I find it likely that BSML will dominate the scene for bioinformatics markup languages for the next year or so, and that its structure will soon be specified by an XML Schema. However, the scope of the Life Sciences Research group (LSR, 5.3.2) is much wider than that of BSML, and I expect that LSR will be able to give useful input during the development of XML Schemas in other areas than sequences. The merging of schemas is especially likely as many organisations are participants in several standardisation co-operations simultaneously.

Binding affinities are the most important external data for a proteo-chemometric analysis and query system (PAQS). In this area the Biomolecular Interaction Network Database (BIND, 5.4.1) seems to provide most data. Furthermore, this data is supported by a publicly available schema. A problem with using BIND as a data source is that the web interface for data access does not allow advanced queries. A further potential problem is that it seems uncertain to what extent future upgrades and improvements will be available free of charge over the Web.

The strength of the mediator/wrapper approach for integrating life science data sources has been shown both by the many projects in academic laboratories (6.1), and by the commercial product DiscoveryLink from IBM (6.2.1). I expect this trend to become even more pronounced with the advent of the SQL/Management of External Data standard (4.4.3).

The database schemas presented and discussed in chapter 8 show that the functional data model of the Amos II system is well suited for the many fairly complex features of the PAQS information domain. For example, the combination of subtyping and function overloading provides a powerful mechanism for polymorphism, which is important both in modelling and in implementation. Furthermore, the vector data type is suitable for representing data series.

The schemas presented for binding experiments and experiment evaluations have been modelled in enough detail for a production mediator system, and they will provide a good basis for the development of a custom markup language. Some important parts of the problem domain, e.g. assays and descriptors, need further consideration.

The use of foreign functions as a means for performing data analysis has been exemplified by experiment evaluations (section 8.8). With the database schemas presented, it will be easy to developed the prototype further in order to allow the complete curve fitting of binding experiments from within the Amos II system. A considerably larger project would be to implement the whole multivariate analysis as foreign functions, and this is not supported by the presented schemas.

References

Printed References

Abiteboul, S., Buneman, P. & Suciu, D. (2000) *Data on the Web: From Relations to Semistructured Data and XML*. San Francisco, California: Morgan Kaufmann.

Ahmed, R., De Smedt, P., Du, W., Kent, W., Ketabchi, M.A., Litwin, W., Rafii, A. & Shan, M.-C. (1991) The Pegasus Heterogeneous Multidatabase System. *IEEE Computer*, 24(12) 19-27.

Aylward, G. & Findlay, T. (1994) *SI Chemical Data*. 3rd ed. Sydney, Australia: John Wiley & Sons.

Bader, G.D., Donaldson, I., Wolting, C., Ouellette, B.F., Pawson, T. & Hogue, C.W. (2001) BIND – The Biomolecular Interaction Network Database, *Nucleic Acids Res.*, 29(1),242-245.

Bader, G.D. & Hogue, C.W. (2000) BIND - a Data Specification for Storing and Describing Biomolecular Interactions, Molecular Complexes and Pathways, *Bioinformatics*, 16(5), 465-477.

Baker, P.G., Goble, C.A., Bechhofer, S., Paton, N.W., Stevens, R. & Brass, A. (1999) An Ontology for Bioinformatics Applications. *Bioinformatics*, 15(6), 510-520.

Banerjee, S. (2000) A Database Platform for Bioinformatics. In *Proc. 26th Int. Conf. Very Large Data Bases* (VLDB 2000), pp 705-710.

Barker, W.C., Garavelli, J.S., Hou, Z., Huang, H., Ledley, R.S., McGarvey, P.B., Mewes, H.-W., Orcutt, B.C., Pfeiffer, F., Tsugita, A., Vinayaka, C.R., Xiao, C., Yeh, L.-S.L. & Wu, C. (2001) Protein Information Resource: A Community Resource for Expert Annotation of Protein Data. *Nucleic Acid Research*, 29(1), 29-32.

Baxevanis, A.D. (2001a) The Molecular Biology Database Collection: an Updated Compilation of Biological Database Resources. *Nucl. Acid Res.* 29(1) 1-10.

Baxevanis, A.D. (2001b) Bioinformatics and the Internet. In *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins*, 2nd ed., (Eds.) Baxevanis, A.D. & Oullette, B.F.F., pp 1-17. New York, N.Y.: John Wiley & Sons.

Baxevanis, A.D. & Oullette, B.F.F. (2001) *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins*. 2nd ed. New York, N.Y.: John Wiley & Sons.

Bellahsene, Z. & Ripoche, H. (2001) An Object-Oriented Database for Managing Genetic Sequences. In *Succeeding with Object Databases: A Practical Look at Today's Implementations with Java and XML*, (Eds.) Chaudhri, A.B. & Zicari, R., 343-356. New York, N.Y.: Wiley.

Berman, H.M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T.N., Weissig, H., Shindyalov, I.N., Bourne, P.E. (2000) The Protein Data Bank. *Nucleic Acids Research*, 28(1), 235-242.

Boman, M., Bubenko Jr., J.A., Johannesson, P., Wangler, B. (1997) *Conceptual Modelling*. London, England: Prentice Hall.

Bouguettaya, A., Benatallah, B. & Elmagarmid, A. (1999) An Overview of Multidatabase Systems: Past and Present. In *Management of Heterogeneous and Autonomous Database*

Systems, (Eds.) Elmagarmid, A., Rusinkiewicz, M. & Sheth, A., 1-32. San Francisco, California: Morgan Kaufmann.

Bourne, P.E., Bermna, H.M., McMahon, B., Watenpaugh, K.D.W., Westbrook, J. & Fitzgerald, P.M.D. (1997) The Macromolecular Crystallographic Information File (mmCIF). *Methods in Enzymology*, 277, 571-590. Avaliable on the Web as http://www.sdsc.edu/pb/cif/papers/methenz.html (2001-06-12).

Buneman, P., Davidson, S.B., Hart, K., Overton, G.C. & Wong, L. (1995) A Data Transformation System for Biological Data Sources. In *Proc. 21st Int. Conf. Very Large Data Bases* (VLDB'95), 158-169.

Cassel, K. & Risch, T. (2001) An Object-Oriented Multi-Mediator Browser. In *2nd Int. Workshop User Interfaces to Data Intensive Systems* (UIDIS2001), p 36-35. Available on the Web as http://www.dis.uu.se/~udbl/publ/goovipaper3.pdf, 2001-08-20.

Cattell, R.G.G., Barry, D.K., Berler, M.D., Eastman, J., Jordan, D., Russel, C., Schadow, O., Stanienda, T. & Velez, F. (2000) *The Object Data Standard: ODMG 3.0*. San Diego, California: Academic Press.

Che, D., Chen, Y. & Aberer, K. (1999) A Query System in a Biological Database. In *Proc. 11th Int. Conf. Scientific and Statistical Database Management* (SSDBM'99), pp 158-167. IEEE Computer Society

Chen, I.-M.A. & Markowitz, V.M. (1995) An Overview of the Object Protocol Model (OPM) and the OPM Data Management Tools. *Information Systems*, 20(5), 393-418. Also available on the Web as http://www.gizmo.lbl.gov/DM_TOOLS/OPM/doc/OPM_3/Overview.ps (2001-02-20).

Connolly, T.M., Begg, C.E. (2002) *Database Systems: A Practical Approach to Design, Implementation, and Management*. 3rd ed. Harlow, England: Addison-Wesley.

Davidson, S.B., Crabtree, J., Brunk, B., Schug, J., Tannen, V., Overton, C. & Stoeckert, C. (2001) K2/Kleisli and GUS: Experiments in Integrated Access to Genomic Data Sources. *IBM Systems Journal*, 40(2), 512-531. Available on the Web as http://www.research.ibm.com/journal/sj/402/davidson.pdf (2001-12-09).

Davidson, S.B., Overton, G.C., Tannen, V. & Wong, L. (1997) BioKleisli: A Digital Library for Biomedical Researchers. *Int. J. on Digital Libraries*, 1(1), 36-53.

De Lean, A., Hancock, A.M., Lefkowitz, R.J. (1982) Validation and Statistical Analysis of Radioligand Binding Data for Mixtures of Pharmacological Receptor Subtypes. *Mol. Pharm.*, 21, 5-16.

Demurjain, S.A & Hsiao, D.K. (1988) Towards a Better Understanding of Data Models Through the Multilingual Database System. *IEEE Trans. Software Engineering*, 14(7), 946-958.

Denning, P.J. (2000) Computer Science. In *Encyclopedia of Computer Science*, 4th ed., (Eds.) Ralston, A., Reilly, E.D. & Hemmendinger, D., 405-419. London, UK: Nature Publishing Group.

Eisenberg, A. & Melton, J. (1999a) SQL:1999, Formerly Known as SQL3. *SIGMOD Record*, 28(1), 131-138. Available on the Web as http://www.acm.org/sigmod/record/issues/9903/standards.pdf.gz (2001-12-04).

Eisenberg, A. & Melton, J. (1999b) SQLJ - Part1: SQL Routines using the Java Programming Language. *SIGMOD Record*, 28(4), 58-63. Available on the Web as http://www.acm.org/sigmod/record/issues/9912/standards.pdf.gz (2001-12-04).

Elmagarmid, A., Du, W. & Ahmed, R. (1999) Local Authonomy and Its Effects on Multidatabase Systems. In *Management of Heterogenous and Autonomous Database Systems*, (Eds.) Elmagarmid, A., Rusinkiewicz, M. & Sheth, A., 33-55. San Francisco, California: Morgan Kaufmann.

Elmagarmid, A., Rusinkiewicz, M. & Sheth, A. (Eds.) (1999*) Management of Heterogenous and Autonomous Database Systems.* San Francisco, California: Morgan Kaufmann.

Elmasri, R., Navathe, S.B. (2000) *Fundamentals of Database Systems*. 3rd ed. Reading, Massachusets: Addison-Wesley.

Embury, S.M. & Gray, P.M.D. (1995) The Declarative Expression of Semantic Integrity in a Database of Protein Structure. In *Proceedings of the Basque International Workshop on Information Technology* (BIWIT'95), (Eds.) Illarramendi, A. & Diaz, O., pp 216-224.

Fenyö, D. (1999) The Bioplymer Markup Language. *Bioinformatics*, 15(4), 339-340.

Fishman, D.H., Beech, D., Cate, H.P., Chow, E.C., Connors, T., Davis, J.W., Derrett, N., Hoch, C.G., Kent, W., Lyngbæk, P., Mahbod, B., Neimat, M.-A., Ryan, T.A. & Shan, M.-C. (1987) Iris: An Object-Oriented Database Management System. *ACM Trans. Office Information Systems*, 5(1) 48-69.

Flodin, S., Orsborn, K. & Risch, T. (1998) Using Queries with Multi-Directional Functions for Numerical Database Applications. In *2nd East-European Symposium on Advances in Databases and Information Systems* (ADBIS'98), p 58-70. Poznan, Poland, September 1998. Available on the Web as http://www.dis.uu.se/~udbl/publ/adbis98.pdf, 2001-08-20.

Fowler, M. (1997) *Analysis Patterns.* Boston, Massachusets: Addison Wesley.

Fowler, M. (2000) *UML Distilled*. 2nd ed. Boston, Massachusets: Addison Wesley.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995) *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Mass.: Addison-Wesley.

Garcia-Molina, H., Ullman, J.D. & Widom, J. (2000) *Database System Implementation*. Upper Saddle River, New Jersy: Prentice Hall.

Grand, M. (1998) *Patterns in Java, Volume 1*. New York, N.Y.: Wiley.

Gray, J.N., Bosworth, A., Layman, A. & Pirahesh, H. (1996) Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-tab, and Sub-totals. In *Proc. 12th Int. Conf. Data Engineering*, pp 152-159. IEEE Computer Society.

Gray, P.M.D. & Kemp, G.J.L. (1990) An OODB with Entity-Based Persistence. *IEEE Colloq. Very Large Knowledge-Based Systems*, pp 4/1-4/4.

Haas, L.M., Miller, R.J., Niswonger, B., Turk Roth, M., Schwarz, P.M. & Wimmers, E.L. (1999) Transforming Heterogeneous Data with Database Middleware: Beyond Integration. *IEEE Data Engineering Bulletin*, 22(1), 31-36.

Haas, L.M., Schwarz, P.M., Kodali, P., Kotlar, E., Rice, J.E. & Swope, W.C. (2001) DiscoveryLink: A System for Integrated Access to Life Sciences Data Sources. *IBM Systems Journal*, 40(2), 489-511. Also available at the Web as http://www.research.ibm.com/journal/sj/402/haas.html (2001-11-29).

Hammer, J., Garcia-Molina, H., Ireland, K., Papakonstantinou, Y., Ullman, J.D. & Widom, J. (1995) Information Translation, Mediation, and Mosaic-Based Browsing in the TSIMMIS System. In *Proc. 1995 ACM SIGMOD Int. Conf. Management of Data* (SIGMOD'95). (Eds.) Carey, M.J. & Schneider, D.A., p 483.

Hammer, J., Garcia-Molina, H., Nestorov, S., Yerneni, R., Breunig, M.M. & Vassalos, V. (1997) Template-Based Wrappers in the TSIMMIS System. In *Proc. 1997 ACM SIGMOD Int. Conf. Management of Data* (SIGMOD'97). (Ed.) Peckham, J., pp 532-535.

Hammer, J. & McLeod, D. (1999) In *Management of Heterogenous and Autonomous Database Systems*, (Eds.) Elmagarmid, A., Rusinkiewicz, M. & Sheth, A., 91-117. San Francisco, California: Morgan Kaufmann.

Harinarayan, V., Rajaraman, A. & Ullman, J.D. (1996) Implementing Data Cubes Efficiently. *Proc. 1996 ACM SIGMOD Int. Conf. Management of Data* (SIGMOD'96), pp 205-216.

Han, J. & Kamber, M. (2001) *Data Mining: Concepts and Techniques.* San Fransisco, California: Morgan Kaufmann.

Haraldsson, A (1993) *Programmering i Lisp* (in swedish). Lund, Sweden: Studentlitteratur.

Hellerstein, J.M., Stonebraker, M. & Caccia, M. (1999) Independent, Open Enterprise Data Integration. *IEEE Data Engineering Bulletin*, 22(1), 43-49.

Hibbert, D.B., & James, A.M. (1987) *MacMillan Dictionary of Chemistry*. London, UK: MacMillan.

Hogue, C.W.V. (2001) Structure Databases. In *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins*, 2nd ed., (Eds.) Baxevanis, A.D. & Oullette, B.F.F., pp 83-109. New York, N.Y.: John Wiley & Sons.

Josifovski, V. (1999) Design, Implementation and Evaluation of a Distributed Mediator System for Data Integration. *Linköping Studies in Science and Technology, Dissertation No. 582*. Linköping, Sweden.

Josifovski, V., Risch, T. (1999) Integrating Heterogeneous Overlapping Databases through Object-Oriented Transformations. In *Proc. 25th Int. Conf. Very Large Data Bases* (VLDB'99), 435-446. Morgan Kaufmann. Available on the Web as http://www.dis.uu.se/~udbl/publ/vldb99.pdf (2001-12-04).

Karsch-Mizrachi, I. & Oulette, B.F.F. (2001) The GenBnk Sequence Database. In *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins*, 2nd ed., (Eds.) Baxevanis, A.D. & Oullette, B.F.F., pp 45-63. New York, N.Y.: John Wiley & Sons.

Kashyap, V. & Sheth, A. (1999) Semantic Similarities Between Objects in Multiple Databases. In *Management of Heterogenous and Autonomous Database Systems*, (Eds.) Elmagarmid, A., Rusinkiewicz, M. & Sheth, A., 57-89. San Francisco, California: Morgan Kaufmann.

Kemp, G.J.L., Angelopoulos, N. & Gray, P.M.D. (2000) A Schema-based Approach to Building a Bioinformatics Database Federation. In *Proc. IEEE Int. Symp. Bio-Informatics and Biomedical Engineering* (BIBE2000), pp 13-20. IEEE Computer Society Press.

Kemp, G.J.L., Dupont, J., & Gray, P.M.D. (1996) Using the Functional Data Model to Integrate Distributed Biological Data Sources. In *Proc. 8th Int. Conf. Scientific and Statistical Database Management* (SSDBM 1996), (Eds.) Svensson, P. & French, J.C., pp176-185.

Kemp, G.J.L., Robertson, C.J., Gray, P.M.D. & Angelopoulos, N. (2000), CORBA and XML: Design Choices for Database Federations. In *Advances in Databases: Proc. BNCOD17*

Conference, (Eds.) Lings, B. & Jeffery, K., pp 191-208. Lecture Notes in Computer Science 1832. Heidelberg, Germany: Springer.

Landers, T. & Rosenberg, R. (1986) An Overview of Multibase. *Distributed Systems*, 2, 391-421.

Lapinsh, M., Prusis, P., Gutcaits, A., Lundstedt, T. & Wikberg, J.E.S. (2001) Development of Proteo-Chemometrics: a Novel Technology for the Analysis of Drug-Receptor Interactions. *Biochimica and Biophysica Acta*, 1525(1-2), 180-190.

Levy, A.Y., Rajaraman, A. & Ordille, J.J. (1996) Querying Heterogeneous Information Sources Using Source Descriptions. In *Proc. 22th Int. Conf. Very Large Data Bases* (VLDB'96), 251-262. Morgan Kaufmann.

Lodish, H., Berk, A., Zipursky, S.L., Matsudaira, P., Baltimore, D. & Darnell, J. (2000) *Molecular Cell Biology*. 4th ed. New York, N.Y.: W.H. Freeman and Company.

Lyngbæk, P. & Kent, W. (1986) A Data Modeling Methodology for the Design and Implementation of Information Systems. In *Proc. 1986 Int. Workshop Object-Oriented Database Syst.*, 6-17.

Maier, D. & Vance, B. (1993) A Call to Order. In *Proc. 12th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems* (PODS'93), pp 1-16. ACM Press.

Malinowski, E. (1991) *Factor Analysis in Chemistry*. 2nd ed. New York, N.Y.: Wiley.

Marcotte, E.M., Xenarios, I. & Eisenberg, D. (2001) Mining literature for protein-protein interactions, *Bioinformatics* 17(4) 359-363.

Melton, J., Michels, J.-E., Josifovski, V., Kulkarni, K.G., Schwarz, P.M. & Zeidenstein, K. (2001) SQL and Management of External Data. *SIGMOD Record*, 30(1), 70-77. Available on the Web as http://www.acm.org/sigmod/record/issues/0103/JM-Sta.pdf.

Mewes H. W., Hani J., Pfeiffer F. & Frishman D. (1998) MIPS: A Database for Protein Sequences and Complete Genomes. *Nucleic Acids Res.*, 26(1), 33-37.

Meyer, B. (1997) *Object-Oriented Software Construction*. 2nd ed. Upper Saddle River, New Jersey: Prentice Hall.

Missier, P., Rusinkiewicz, M. & Jin, W. (1999) Multidatabase Languages. In *Management of Heterogenous and Autonomous Database Systems*, (Eds.) Elmagarmid, A., Rusinkiewicz, M. & Sheth, A., 175-216. San Francisco, California: Morgan Kaufmann.

Munson, P.J., Rodbard, P. (1980) A Versatile Computerized Approach for the Characterization of Ligand Binding Systems. *Anal. Biochem.*, 107, 220-239.

Murray-Rust, P., Rzepa, H.S., Wright, M. & Zara, S. (2000) A Universal Approach to Web-Based Chemistry using XML and CML, *ChemComm*, 2000, 1471-1472.

Novak, G. S., Jr. (1995) Conversion of Units of Measurement. *IEEE Trans. Software Engineering*, 21(8), 651-661.

O'Neil, P. & O'Neil, E. (2001) *Database - Principles, Programming, and Performance*. 2nd ed. San Fransisco, Calif.: Morgan Kaufmann.

Ostell, J.M., Wheelan, S.J. & Kans, J.A. (2001) The NCBI Data Model. In *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins*, 2nd ed., (Eds.) Baxevanis, A.D. & Oullette, B.F.F., pp 19-43. New York, N.Y.: John Wiley & Sons.

Özsu, M.T. & Valduriez, P. (1999) *Principles of Distributed Database Systems.* 2nd ed. Upper Saddle River, New Jersey: Prentice Hall.

Paton, N.W., Khan, S.A., Hayes, A., Moussouni, F., Brass, A., Eilbeck, K., Goble, C.A., Hubbard, S.J. & Oliver, S.G. (2000) Conceptual modelling of genomic information. *Bioinformatics*, 16(6), 548-557.

Paton, N.W. (2001) Experience Using the ODMG Standard in Bioinformatics Applications. In *Succeeding with Object Databases: A Practical Look at Today's Implementations with Java and XML*, (Eds.) Chaudhri, A.B. & Zicari, R., 329-341. New York, N.Y.: Wiley.

Pitoura, E., Bukhres, O. & Elmagarmid, A. (1995) Object Orientation in Multidatabase Systems. *ACM Computing Surveys*, 27(2), 141-195.

Pitt, V. H. (1977) *The Penguin Dictionary of Physics.* Harmondsworth, England: Penguin Books.

Prusis, P. (2001) Modelling of Melanocortin Receptors and Their Ligands. *Comprehensive Summaries of Uppsala Dissertations from the Faculty of Pharmacy 249*. Uppsala, Sweden: Acta Universitatis Upsaliensis.

Prusis, P., Muceniece, R., Andersson, P., Post, C., Lundstedt, T. & Wikberg, J.E.S. (2001) PLS Modeling of Chimeric MS04/MSH-peptide and MC_1/MC_3-Receptor Interactions Reveals a Novel Method for the Analysis of Ligand-Receptor Interactions. *Biochimica and Biophysica Acta*, 1544(1-2), 350-357.

Rezende, F.F. & Hergula, K. (1998) The Heterogeneity Problem and Middleware Technology: Experiences with and Performance of Database Gateways. In *Proc. 24th Int. Conf. Very Large Data Bases* (VLDB'98), pp 146-157.

Risch, T. & Josifovski, V. (2001) Distributed Data Integration by Object-Oriented Mediator Servers. *Concurrency and Computation: Practice and Experience*, 13(11), 933-953. Available on the Web as http://www.dis.uu.se/~udbl/publ/ddiooms.pdf, 2001-12-04.

Siepel, A.C., Tolopko, A.N., Farmer, A.D., Steadman, P.A., Schilkey, F.D., Perry, B.D. & Beavis, W.D. (2001) An Integration Platform for Heterogeneous Bioinformatics Software Components. *IBM Systems Journal*, 40(2), 570-591.

Silberberg, M. (1996) *Chemistry.* St. Louis, Missouri: Mosby-Year Book.

Shipman, D.W. (1981) The Functional Data Model and the Data Language DAPLEX. *ACM Trans. Database Systems*, 6(1), 140-173.

Stonebraker, M. & Brown, P. (1999) *Object-Relational DBMSs: Tracking the Next Great Wave.* 2nd ed. San Fransisco, California: Morgan Kaufmann.

Tomasic, A., Raschid, L. & Valduriez, P. (1998) Scaling Access to Heterogeneous Data Sources with DISCO. *IEEE Trans. Knowledge and Data Engineering*, 10(5), 808-823.

Tork Roth, M. & Schwarz, P. (1997) Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. In *Proc. 23rd Int. Conf. Very Large Databases* (VLDB'97), pp 266-275.

Turner, D.B., Tyrell, S.M. & Willet, P. (1997) Rapid Quantification of Molecular Diversity for Selective Database Acquisition. *J. Chem. Inf. Comput. Sci.*, 37(1), 18-22.

Westbrook, J.D. & Bourne, P.E. (2000) STAR/mmCIF: An Ontology for Macromolecular Structure. *Bioinformatics*, 16(2), 159-168.

Wiederhold, G. (1990) Future Architectures for Information Processing Systems. In *Proc. PARBASE-90 (First Int. Conf. Databases, Parallel Architectures and their Applications)*. (Eds.) Rishe, N., Navathe, S. & Tal, D., 1-20.

Wiederhold, G. (1992) Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3), 38-49.

Wiederhold, G. & Genesereth, M. (1997) The Conceptual Basis for Mediation Services. *IEEE Expert*, 12(5), 38-47.

Wikberg, J. (2001). *BindAid: Ligand Binding Data Analysis for the Macintosh Computer*. Program manual for BindAid v 1.0. Jarl Wikberg, Sigtuna, Sweden.

Wilkinson, K., Lyngbæk, P. & Hasan, W. (1990) The Iris Architecture and Implementation. *IEEE Trans. Knowledge and Data Engineering*, 2(1) 63-75.

Woolf, B. (1997) Null Object. In *Pattern Languages in Program Design 3*. (Eds.) Martin, R., Riehle, D. & Buschmann, F., pp 5-18. Reading, Mass.: Addison-Wesley.

Xenarios, I., Fernandez, E., Salwinski, L., Duan, X.J., Thompson, M.J., Marcotte, E.M. & Eisenberg, D. (2001) DIP- The Database of Interacting Proteins: 2001 Update, *Nucl. Acids. Res.*, 29(1), 239-241.

Xenarios, I., Rice, D.W., Salwinski, L., Baron, M.K., Marcotte, E.M. & Eisenberg, D. (2000) DIP- The Database of Interacting Proteins, *Nucl. Acids. Res.*, 28(1), 289-291.

Private Communications

Andersson, Arne, Computing Science Department, Uppsala University. Lecture on Data Mining, 2000-11-25.

Odelstad, Jan, Department of Computer Science, Gävle University, 2001-12-13.

Petrini, Johan, *Accessing Web Forms from an Object-Relational DBMS*, M.Sc. Thesis presentation, 2001-10-26.

Rehn, Catharina, Karolinska Institutet Library. Lecture on Data Mining, 2000-11-30.

SRS Relational (2001) Overhead copies from the SRS User Group Meeting 2001-03-21. Lion Bioscience.

Thelen, Frank, Poet Software GmbH. XML - Problem or Solution? In *Poet Developer Network for FastObjects*, No 3, 2001.

Web References

AceDB Home Page, http://www.acedb.org/, 2001-12-12.

aeFDM: FDM implementation of the ArrayExpress Schema, http://www.csd.abdn.ac.uk/~gjlk/microarray/daplex_schema.html, 2001-12-17.

Array XML (AXML), http://www.xml.org/xml/zapthink/std409.html, 2001-12-17.

ArrayExpress Schema, Model and Documentation, http://www.ebi.ac.uk/arrayexpress/Schema/schema.html, 2001-12-17.

ASN.1 Information Site, http://asn1.elibel.tm.fr/en/index.htm, 2001-12-12.

BIND - The Biomolecular Interaction Network Database, http://www.binddb.org, 2001-12-10.

Bloch, J. (2001) *Substitutes for Missing C Constructs*, http://developer.java.sun.com/developer/Books/shiftintojava/page1.html, 2001-08-15.

Brandani, S. (1998) *Multi-database Access from Amos II using ODBC*. In *Linköping Electronic Press*, 3 (19). http://www.ep.liu.se/ea/cis/1998/019, 2001-12-13.

BSML - An Emerging Industry Standard for Linking Genomic Sequences to Biological Function, http://www.bsml.org/, 2001-12-14.

Butt, A. (2001) *Proteomics Primer*. http://www.spectroscopynow.com/Spy/basehtml/SpyH/1,7455,0-1-2-0-0-news_detail-0-110,00.0.html, 2001-06-11.

Chemical Markup Language, http://ww.xml.cml.org, 2001-12-14.

CORBA at EBI, http://corba.ebi.ac.uk/index.html, 2001-12-06.

Cover, R. 2001. *Microarray Markup Language (MAML)*, http://www.oasis-open.org/cover/maml.html, 2001-12-16.

DiscoveryLink, http://www-3.ibm.com/solutions/lifesciences/discoverylink.html, 2001-12-18.

Elin, D. & Risch, T. (2001) *Amos II Java Interfaces*. Available as a part of the Amos II distribution: http://www.dis.uu.se/~udbl/amos/, 2001-08-20.

Entrez Home, http://www.ncbi.nlm.nih.gov/Entrez/, 2001-12-18.

Entrez PubMed, http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=PubMed, 2001-12-18.

Establishing a Public Repository for DNA Microarray-Based Gene Expression Data, http://www.ebi.ac.uk/arrayexpress/News/news.html, 2001-12-17.

ExPASy - SWISS-PROT and TrEMBL, http://us.expasy.org/sprot/, 2001-12-17.

Flodin, S., Josifovski, V., Katchaounov, T., Risch, T., Sköld, M., Werner, M. (2000) *Amos II User's Manual (June 23, 2000)*. http://www.dis.uu.se/~udbl/amos/doc/amos_users_guide.html, 2001-08-20.

Garlic, http://www.almaden.ibm.com/cs/garlic.html, 2001-12-18.

Genome Information Management System, http://img.cs.man.ac.uk/gims/, 2001-12-17.

GPCRDB, http://www.gpcr.org/, 2001-12-10.

GPCR Query, http://www.darmstadt.gmd.de/~gpcrdb/, 2001-12-10.

Haas, L.M., Schwarz, P.M., Kodali, P., Kotlar, E., Rice, J.E. & Swope, W.C. (2001) *DiscoveryLink: A System for Integrated Access to Life Sciences Data Sources*. http://www.research.ibm.com/journal/sj/402/haas.html (2001-11-29). Printed version: *IBM Systems Journal*, 40(2), 489-511.

Hamilton, B. (1996) *A Compact Representation of Units*, http://www.hpl.hp.com/techreports/96/HPL-96-61.ps, 2001-07-13.

Human Proteomics Initiative, http://www.expasy.org/sprot/hpi/, 2001-12-06.

I3C Demo at BIO2001, http://www.i3c.org/Bio2001/i3c_demo_final.htm, 2001-12-14.

I3C Home Page, http://www.i3c.org/, 2001-12-14.

IBM and MDS Proteomics Alliance Aims to Speed Drug Development, http://www.mdsproteomics.com/default.asp?qID=8&qType=PressDisplay, 2001-12-10.

IBM Life Sciences, http://www-3.ibm.com/solutions/lifesciences/, 2001-12-13.

IBM Life Sciences Framework, http://www-3.ibm.com/solutions/lifesciences/pdf/IBM_LSFramework_v2.pdf, 2001-12-13.

INCOGEN VIBE, http://www.incogen.com/vibe/, 2002-01-10.

Interact - A Protein-Protein Interaction database, http://www.bioinf.man.ac.uk/resources/interact.shtml, 2001-12-10

Introduction to LSR, http://www.omg.org/homepages/lsr, 2001-12-06.

James, C.A., Weininger, D. & Delany, J. (2000) *Daylight Theory - Fingerprints*. http://www.daylight.com/dayhtml/doc/theory/theory.finger.html, 2001-04-11.

Janicijevic, I. & Fowler, M. (2001) *UML Diagrams for Chapter 3 of Analysis Patterns*. http://martinfowler.com/apsupp/apchap3.pdf, 2001-08-15.

Java Architecture for XML Binding , http://java.sun.com/xml/jaxb/, 2002-01-17.

java.lang.reflect Class Method (2001) http://java.sun.com/products/jdk/1.2/docs/api/java/lang/reflect/Method.html, 2001-08-25.

Kappler, M.A. & Delany, J. (2001) *Daylight / Oracle Cartridge*. http://www.daylight.com/meetings/emug01/Kappler/DayCart/cartridge.html, 2001-11-27.

Kemp, G.J.L., Fothergill, J., Gray, P.M.D. & Angelopoulos, N. (2001), *Development of a Mediator to Integrate Access to Databases in Molecular Biology*. Abstract for Grantholders Workshop, February 2001. http://www.csd.abdn.ac.uk/~gjlk/research/medreport4.html, 2001-12-08.

Kent, W., Janowski, S. L., Hamilton, B. & Hepner, D. (1996) *Measurement Data*. http://www.hpl.hp.com/techreports/96/HPL-96-73.ps, 2001-07-13.

Larmouth, J. (2001) *ASN.1 is Reaching Out*, http://www.asn1.org/Jpaper.pdf , 2001-12-12.

Linking Biological Databases using the Common Object Request Broker Architecture, http://corba.ebi.ac.uk/RHdb/EUCORBA/, 2001-12-06.

McArthur, D.C. (2001) An extensible XML Schema definition for automated exchange of protein data: PROXIML (PROtein eXtensIble Markup Language), http://www.cse.ucsc.edu/~douglas/proximl/, 2001-12-14.

Markowitz, V.M., Chen, I.A., Kosky, A.S. & Szeto, E. (2001) *Facilities for Exploring Molecular Biology Databases on the Web: A Comparative Study*, http://gizmo.lbl.gov/DM_TOOLS/OPM/WebInt/WebInt.ps, 2001-12-13.

Murray-Rust, P. & Rzepa, H.S. (2001) *Chemical Markup Language. A Position Paper*. http://www.xml-cml.org/information/position.html, 2001-12-14.

NCBI Toolbox, http://www.ncbi.nlm.nih.gov./IEB/ToolBox/index.cgi, 2001-12-12.

PIR-International Protein Sequence Database, http://pir.georgetown.edu/pirwww/dbinfo/pirpsd.html, 2001-12-16.

Relibase, http://www.ccdc.cam.ac.uk/prods/relibase, 2001-12-10.

Resource Description Framework (RDF), http://www.w3.org/RDF/, 2001-12-17.

Risch, T., Josifovski, V. & Katchatounov, T. (2000) *Amos II Concepts (June 23, 2000)*. http://www.dis.uu.se/~udbl/amos/doc/amos_concepts.html, 2001-08-20.

SigmaPlot. Exact Graphs for Exact Science, http://www.spssscience.com/SigmaPlot/index.cfm, 2001-08-26.

SRS 6, http://srs6.ebi.ac.uk/, 2001-12-17.

SRS 6 Extensions, http://www.lionbioscience.com/htm/c_1/content_c_1_1_2.htm, 2001-04-06.

SRS Technology, http://www.lionbioscience.com/htm/c_1/content_c_1_1_1.htm, 2001-04-06.

Structure and Design of ArrayExpress Database, http://www.ebi.ac.uk/arrayexpress/Design/design.html, 2001-12-17.

TAMBIS, http://img.cs.man.ac.uk/tambis/index.html, 2001-12-06.

The ArrayExpress Database, http://www.ebi.ac.uk/arrayexpress/, 2001-12-17.

The Binding Database, http://www.bindingdb.org, 2001-04-17.

The BIOML home page, http://www.bioml.com/BIOML/index.html, 2001-12-14.

The Genome Database, http://www.gdb.org, 2001-04-20.

The maxdSQL Database, http://bioinf.man.ac.uk/microarray/maxd/maxdSQL/index.html, 2001-12-17.

The OPM Project, http://gizmo.lbl.gov/opm.html, 2001-12-17.

The RCSB Protein Data Bank, http://www.rcsb.org/pdb/, 2001-12-17.

The XEBML Project, http://www.ebi.ac.uk/xembl, 2001-12-06.

Welcome to AGAVE, http://www.agavexml.org/, 2001-12-14.

XML Metadata Interchange Format (XMI), http://www-4.ibm.com/software/ad/library/standards/xmi.html, 2001-12-16.

Appendix A : Abbreviations

The following list includes some abbreviations which occur in several places in the text, figures, or code excerpts. Many abbreviations which only occur in a single paragraph or short section (together with an explanation) are not included.

Amos	Active Mediator Object System
API	application programming interface
BA, ba	binding affinity
BIND	Biomolecular Interaction Network Database
BSML	Bioinformatic Sequence Markup Language
CBE	concentration binding experiment
CDS	concentration data series
CE	ChemicalEntity
CML	Chemical Markup Language
conc	concentration
DB	database
DBMS	database management system
DBS	database system
DDBS	distributed database system
DDL	data definition language (with mmCIF: data dictionary language)
DML	data manipulation language
DPM, dpm	disintegrations per minute
DS, ds	data series (also data source in Appendix G)
DTD	data type definition (in connection with XML)
EBI	European Bioinformatics Institute
ee	experiment evaluation
fm	fit model
fp	fit parameter(s)
GUI	graphical user interface
I3C	Interoperable Informatics Infrastructure Consortium
JDBC	"Java database connectivity", officially not an abbreviation
m	metre
M	molar, mol/dm3, a unit for concentrations of solutions
MED	Management of External Data (in connection with SQL)
NCBI	National Center for Biotechnology Information
nM	nanomolar (10-9 M), a unit for concentrations of solutions
ODBC	open database connectivity
ODMG	Object Data Management Group
OID	object identifier
OMG	Object Management Group
OO	object-oriented, object-orientation
OODBMS	object-oriented database management system
OQL	Object Query Language
ORDBMS	object-relational database management system
PAQS	Proteo-chemometric Analysis and Query System
PDB	Protein Data Bank, also a popular file format
PIR	Protein Information Resource
RDBMS	relational database management system
SI	Système International (d'Unités)
SQL	Structured Query Language
SRS	Sequence Retrieval System
TBE	time binding experiment
UML	Unified Modeling Language
wm	weighting model
wp	weighting parameter(s)
XML	eXtensible Markup Language

Appendix B : Examples from the Prototype

In this Appendix a short demonstration of the implemented schema is given. For a larger demonstration, see the electronic supplementary material: "prot1_observationData.amosql" (some binding affinity data from the literature), "load_appendix45.amosql" (load data of the BindAid manual, appendices 4 & 5), and "demo_queries_appendix45_part[1-7].amosql" (queries over the loaded data).

B.1 A Competition Binding Experiment

Here we will see how a competition binding experiment may be stored in the database. The data is taken from the BindAid manual (Wikberg 2001), appendix 4. The corresponding graphical representation of the schema is given in Figure 35.

```
/* The binding assay */
create BindingAssay(name) instances
   :app4_ass1 ("Assay of BindAid Appendix4");
add description(:app4_ass1) = "Cerebral cortex membranes";
add bindingEntity(:app4_ass1) = :alfa2A;

/* One concentration binding experiment,
   a non-masked competition curve */
create ConcBindingExperiment(name, cbeType) instances
   :app4_exp1 ("curve 1, p 57", :competition);
set description(:app4_exp1) =
   "non-masked competition curve for BRL44408";
add_fixed_ligand(:app4_exp1, :h3_mk912 , 0.326, "nM");
add curves(:app4_ass1)=:app4_exp1;

/* Conversion from dpm units to nM according to page 57 */
set :temp = times(div(1.0, 13486.0), 1.0e-9);
set :dpm_13486 = createUnit("dpm (13486)", :temp, :molarity);

/* Two dataseries per curve,
   one independent (varying) and one dependent (bound) */
create ConcDataSeries(ligand, unit, property) instances
   :ds1v (:brl44408, :nM, :conc);
set values(:ds1v) =
   { 100000.0, 31645.6, 10014.4, 3169.1, 1002.9, 317.37,
     100.43, 31.783, 10.058, 3.1828, 1.0072, 0.3187 };
create ConcDataSeries(ligand, unit, property ) instances
   :ds1b (:h3_mk912, :dpm_13486, :conc );
set values(:ds1b) =
   { 31.0, 31.0, 36.0, 46.0, 55.0, 89.0,
     115.0, 137.0, 186.0, 260.0, 305.0, 328.0 };
add varying(:app4_exp1) = :ds1v;
add bound(:app4_exp1)   = :ds1b;

/* Get a handle to the desired binding experiment */
select e into :e41
   from Experiment e where name(e) = "curve 1, p 57";

/* Demo queries with answers */
count(allData(:e41)); /* How many data series in the experiment? */
answer: 2
consistent(:e41); /* All data series are equally long? */
answer: TRUE

name(all_ligands(:e41)); /* Ligands involved in this experiment */
answer: "3H-MK912"
        "BRL44408"
```

```
/* How many data points are there in the bound radioligand
   dataseries? And how many of these are non-valid? */
select length(ds) from DataSeries ds where ds=bound(:e41);
answer: 12
count(nonvalids(bound(:e41)));
answer: 0
name(unit(bound(:e41))); /* Unit of bound data series */
answer: "dpm (13486)"
convert_series(bound(:e41), get_unit("nM")); /* Series in nM */
answer: { 0.00229868, 0.00229868, 0.00266943, 0.00341094, 0.0040783,
          0.00659944, 0.00852736, 0.0101587,  0.0137921,  0.0192793,
          0.022616,   0.0243215 }
```

B.2 An Experiment Evaluation

The schema for experiment evaluations is described in section 8.8. The following demonstration is part of "load_appendix45.amosql". It certainly is too tedious to type in this much to store a single curve fit, but if a GUI is used only the numbers need be entered. Two even better solutions are (i) to read the values directly from some output of the fit program, and (ii) to invoke the fit program directly from the Amos II system, and let it return the results.

```
/* Create an instance of ExperimentEvaluaton and link it to three
   binding experiments and a fit method */
create ExperimentEvaluation instances :app4_expeval1;
set experiments(:app4_expeval1) = :app4_exp1;
add experiments(:app4_expeval1) = :app4_exp2;
add experiments(:app4_expeval1) = :app4_exp3;
set method(:app4_expeval1) = :bindfit_calc;
set name(:app4_expeval1) =
    "Bindfit calculation from BindAid manual, " +
    "Appendix4, p59 (one binding site)";

/* Create the fit parameters */
setupFitParameters( :app4_expeval1 );
assertEquals( 7, numberOfFitParameters(:app4_expeval1) );

/* Get hold of parameters and set constraints/
set :k11 = getParameter(:app4_expeval1,:affinity,:alfa2A,:brl44408);
set :k21 = getParameter(:app4_expeval1,:affinity, :alfa2A,
                        :rauwolscine);
set :k31 = getParameter(:app4_expeval1,:affinity,:alfa2A,:h3_mk912);
set :n1  = getParameter(:app4_expeval1,:nonspecbind,:brl44408);
set :n2  = getParameter(:app4_expeval1,:nonspecbind,:rauwolscine);
set :n3  = getParameter(:app4_expeval1,:nonspecbind,:h3_mk912);
set :r1  = getParameter(:app4_expeval1,:conc,:alfa2A);
setFitConstraintFloating(:r1, 0.1);
setFitConstraintFloating(:k11, 0.01);
setFitConstraintFloating(:k21, 0.1);
setFitConstraintConstant(:k31, 0.74);
setFitConstraintFloating(:n1, 0.01);
setFitConstraintShared(:n2, :n1);
setFitConstraintShared(:n3, :n1);

/* Fit results */
set sumOfSquaredResidues(:app4_expeval1) = 31.98;
setFloatingResult( :r1,  0.1150,   0.004694 );
setFloatingResult( :n1,  0.008876, 0.0009101 );
setFloatingResult( :k11, 0.03448,  0.005875 );
setFloatingResult( :k21, 0.4801,   0.1056);

/* Demo queries with answers */
name(experiments(:app4_expeval1)); /* Experiments in evaluation */
answer: "curve 3, p 58"
        "curve 2, p 58"
        "curve 1, p 57"
```

```
name(model(method(:app4_expeval1))); /* Fit model used */
answer: "Bindfit"
/* Number of fit parameters, and distribution over types */
numberOfFitParameters(:app4_expeval1);
answer: 7
select name(p), count(getParameter(:app4_expeval1, p))
   from Property p
   where count(getParameter(:app4_expeval1, p)) > 0;
answer: <"affinity constant",3>
        <"concentration",1>
        <"ligand non-specific binding parameter",3>
/* What were the results of the fitted [floating] affinities?
   binding site, ligand, value, unit, estimated error */
select name(bindingSite(fp)), name(ligand(fp)), value(fp),
       name(unit(fp)), error(fp)
   from FitParameter fp
   where isFloating(fp) and fp = getParameter(:app4_expeval1,
                                 get_property("affinity constant"));
answer:
   <"alfa2A adrenoceptor","Rauwolscine",0.4801,"(nM)^-1",0.1056>
   <"alfa2A adrenoceptor","BRL44408",0.03448,"(nM)^-1",0.005875>
```

B.3 Search for Binding Affinities

The following demonstration assumes some data has been stored in the database, e.g. through the scripts "prot1_observationData.amosql". It is further assumed that some experiment evaluations are stored, e.g. the one from section B.2.

```
/* How many binding affinities are stored as observations and
   how many are stored as results of experiment evaluations? */
set :aff = get_property("affinity constant");
count( allObs( :aff ) );
answer: 8
count( getParameters( :aff ) ); /* all: constant, floating, . . . */
answer: 3
count( isFloating(getParameters(:aff)) ); /* only floating */
answer: 2

/* Focus on the 'alfa2A adrenoceptor' */
set :a2A = getCE('alfa2A adrenoceptor');
count( allObs( :aff, :a2A ) );
answer: 4
list_affinities( :a2A );
answer: <"alfa2A adrenoceptor","Risperidone"," 23.0 (nM)^-1 ">
        <"alfa2A adrenoceptor","BRL44408"," 5.68 (nM)^-1 ">
        <"alfa2A adrenoceptor","BDF8933"," 2.0 (nM)^-1 ">
        <"alfa2A adrenoceptor","3H-MK912"," 0.74 (nM)^-1 ">
count( getFittedAffinities( :a2A ) );
answer: 2
getLigandsAndFittedAffinitiesForSite( :a2A );
answer: <"Rauwolscine",0.4801,"(nM)^-1",0.1056>
        <"BRL44408",0.03448,"(nM)^-1",0.005875>
/* Where does the alfa2A:Risperidone value come from? */
select description(reference(o))
   from Observation o
   where o = data(:aff, :a2a, getCE("Risperidone"));
answer: "Schotte A, Janssen PF, Gommeren W, Luyten WH, Van Gompel P,
         Lesage AS, De Loore K, Leysen JE.:
         'Risperidone compared with new and reference antipsychotic
         drugs: in vitro and in vivo receptor binding.'
         Psychopharmacology (Berl) 1996 Mar;124(1-2):57-73."
         /* Taken from PDSP drug database on the Web */
```

Appendix C : Vector and Bag Manipulations

This Appendix describes a few vector and bag operations implemented as foreign functions. Only the Amos II functions and excerpts of the Java code will be given here. For full Java code, see supplementary material (Prot1Vector.java, Prot1BagOperations.java). For an explanation of the Amos II - Java interface, see Elin and Risch 2001.

These functions are intended for use with PAQS prototypes only. I believe these operations, and several more vector and matrix[266] operations should be included in the Amos II system, but it would probably be preferably if they were implemented in Lisp or C and integrated into the Amos II core distribution.

C.1 Vector Arithmetics

A vector of reals may be scaled, i.e. each of its elements multiplied by a real number[267], by means of the function `scale_vector`. This operation is used for converting data series from one unit to another.

```
create function scale_vector( vector of real v, real sf ) ->
  vector of real
  as select scale_vector_foreign( v, sf );

create function scale_vector_foreign( vector of real v, real sf ) ->
  vector of real as foreign "JAVA:Prot1Vector/scaleVector";
```

Two (equal length) vectors of real numbers may be multiplied element by element, and the result put in a new vector.

```
create function multiply_vectors( vector v1, vector v2 ) ->
 vector of real
 as select multiply_vectors_foreign( v1, v2 );

create function multiply_vectors_foreign( vector v1, vector v2 ) ->
  vector as foreign "JAVA:Prot1Vector/multiplyVectors";
```

A bag of vectors may also be averaged, element by element:

```
create function average_vectors( bag b ) ->
 vector of real
 as select average_vectors_foreign( vectorof(b) );

create function average_vectors_foreign( vector v ) ->
  vector as foreign "JAVA:Prot1Vector/averageVectors";
```

Other vector operations which would be useful in scientific and engineering applications are the inner product (or dot product) and the cross product (or vector product).

[266] A nice example of the use of matrices for a database system in the domain of finite element analysis is given by Flodin, Orsborn and Risch (1998). In this work the matrix operations (e.g. various matrix-vector multiplications) were implemented as foreign functions in C. These function definitions are, however, not included in the Amos II distribution, and there seems to be no mentioning of them in the documentation.

[267] In vector calculus terminology this is the *scalar product*.

C.2 Bag Operations

Union, intersection, and difference of bags were also implemented as foreign functions in Java. Bag difference was used in assertions, see Appendix D:

```
create function bagDifference( bag a, bag b ) -> Object
  as select in( foreignBagDifference( vectorof(a), vectorof(b) ) );
create function foreignBagDifference( vector a, vector b ) ->
  vector as foreign "JAVA:Prot1BagOperations/bagDifference";

create function bagUnion( bag a, bag b ) -> Object
  as select in( foreignBagUnion( vectorof(a), vectorof(b) ) );
create function foreignBagUnion( vector a, vector b ) -> vector
  as foreign "JAVA:Prot1BagOperations/bagUnion";

create function bagIntersection( bag a, bag b ) -> Object
  as select in( foreignBagIntersection(vectorof(a),vectorof(b)) );
create function foreignBagIntersection(vector a,vector b) -> vector
  as foreign "JAVA:Prot1BagOperations/bagIntersection";
```

C.3 Data Series Averaging

The averaging of data series was described in section 8.2.5. The operation is complicated by the fact that a data series may have one or several non-valid data points, which are not to be included in the averaging. E.g., suppose data series A has the values v_A={1.0, 2.2, 3.0, 44.0, 5.1} and data series B the values v_B={1.0, 1.8, 3.2, 4.0, 4.9}. Then it is highly likely that the fourth data point of v_A is in error, and that the correct average should be {1.0, 2.0, 3.1, 4.0, 5.0}[268]. Such nonvalid points are handled by a bag `nonvalids` (see Figure 28).

The values for a new (averaged) data series may be obtained with the function `average_dataseries(bag b, Unit u) -> vector of real`. Each data series in the bag `b` is converted to the desired unit `u`, and then the function `average_vectors_nonvalids` is invoked with an argument consisting of a bag of tuples `<"vector of converted values","vector of nonvalid point indexes">`. This bag of tuples is converted to a vector and the foreign Java function `Prot1Vector.averageValidVectors` is invoked[269]. The structure of the vector shipped out to the foreign function thus is the following:

```
vector
   vector_for_dataseries1
      vector1_values
      vector1_nonvalids
   vector_for_dataseries2
      vector2_values
      vector2_nonvalids
   vector_for_dataseries3
      vector3_values
      vector3_nonvalids
   ...
```

The algorithm in `Prot1Vector.averageValidVectors` will only take the average of the data series, future enhancements could allow the calculation of variance and other measures of spread. A very coarse description of `Prot1Vector.averageValidVectors` is as follows:

[268] Thus, the fourth element of the result is an "average" of one value only. A peculiarity of the implemented foreign function is that the result vector will have the value -999.999 in any position where *all* of the input vectors have a nonvalid elements. Possibly, it would be better to return null.

[269] This fairly complicated solution was chosen in order to ascertain that the Java method will be able to match the non-valid indexes to the correct vector. Another constraint is that the Amos II interface to Java foreign functions requires a single vector (containing *all* arguments) to be shipped out.

```
1. Read input into local Java variables
2. Check that all vectors are of equal length
3. Sum up all input vectors to a result vector, element by element
4. Check that nonvalid points are correctly defined,
   and remove them from the sum
5. Divide each element of the result vector by number of points used
6. Put back into result tuple. Emit
```

The three basic functions for data series averaging are:

```
create function average_dataseries(bag b, Unit u)
-> vector of real
as select average_vectors_nonvalids( select convert_series(ds, u),
                                     vectorof(nonvalids(ds) )
   from DataSeries ds
   where ds = in(b) );

create function average_vectors_nonvalids( bag b )
-> vector of real
as select average_vectors_nonvalids_foreign( vectorof(b) );

create function average_vectors_nonvalids_foreign( vector v )
-> vector
as foreign "JAVA:Prot1Vector/averageValidVectors";
```

Finally, as described in section 8.2.5 a few of useful utility functions have been implemented:

- Function `average2nM` takes a bag of `DataSeries` as argument and calculates their average in nM units:

```
create function average2nM( bag b ) ->
  vector of real
  as select average_dataseries(
     ( select ds from ConcDataSeries ds where ds = in(b) ),
     get_unit("nM") );
```

- `average_bound(BindingExperiment,ChemicalEntity)` averages all "bound" data series for a specified `BindingExperiment` and `ChemicalEntity`:

```
create function average_bound( BindingExperiment e,
                               ChemicalEntity ligand ) ->
  vector of real
  as select average2nM( select ds from ConcDataSeries ds
                        where ds=bound(e) and ligand=ligand(ds) );
```

- `average_bound(BindingExperiment)` averages all "bound" data series for each distinct ligand in a specified `BindingExperiment`:

```
create function average_bound( BindingExperiment exp ) ->
  <charstring, vector of real>
  as select name(ligand), average_bound( exp, ligand )
     from ChemicalEntity ligand
     where ligand = bound_ligands( exp );
```

Appendix D : Tests and Assertions

During the implementation of a growing schema it is important that tests are performed frequently. Tests help the developer to find out when a feature starts working, and also when it stops working (e.g. due to unintended side effects).

For example, when the types `Quantity`, `Unit`, and `UnitType` are implemented (see sections 8.1.10, 8.1.11), it is necessary to check that the unit conversions and quantity arithmetics give correct results, and this should preferably be checked after each change affecting these types. For many types, e.g. `DataSeries` and `BindingExperiment`, the main purpose of tests are to ascertain that more or less complicated sequences of function invocations behave as assumed.

The very simplest tests are those which print out a message or an intermediate result. However, with this method of visual inspection it soon becomes tedious to go through the outputs to make sure the results are correct, and in practise only a few tests are run. A better solution would be a silent and automatic test, which is easy to run and only reports errors. Such tests exist for application development in programming languages, e.g. the JUnit framework[270] for development in Java.

The tests made during the development of the schemas in this Thesis follows the philosophy of JUnit, and also have taken over some structure from that package. The JUnit framework as such is not used, however, and the testing is performed in AMOSQL scripts, not in a Java class inheriting from JUnit's TestCase class. All test functionality is implemented in "prot1_assert.amosql" and "Prot1Assert.java", which in turn use "prot1_utility.amosql" and "Prot1BagOperations.java".

Let us now return to the types `Quantity`, `Unit`, and `UnitType` in order to exemplify the testing. The following (edited) excerpt of "prot1_quantityExamples.amosql" contains several kinds of tests:

```
create Quantity (amount, unit) instances
  :q1 (2.0, :hours), :q2 (120.0, :minutes);

assertEquals( 2.0, amount(:q1) );
assertEquals( "hours", name(unit(:q1)) );
/* unit conversion */
assertEquals( 120.0, convert(:q1, :minutes) );
assertEquals( 7200.0, convert(:q1, "seconds") );
/* quantity arithmetics */
assertEquals( 4.0, amount(plus(:q1,:q2,:hours)) );

/* equality tests on bags require another function */
assertBagEquals( bag(:kelvin,:celsius), units(:temperatureUnit) );

set :u1a = createUnit("u1",1.1,:massUnit);
/* unnecessary unit creation, instead :u2a is assigned to :u1a */
set :u2a = createUnit("u2",1.1,:massUnit);
assertEquals( 1.1, scaleFactor(:u2a) );
assertEquals( :u1a, :u2a );

/* All previous assertions are passed */
/* The following would throw an AssertionFailedError,
   and interrupt the script
assertEquals( :seconds, :hours ); */
```

As may be seen in the examples, `assertEquals` is overloaded, and can take either two `Objects` as argument, or a message string (which is printed if the assertion fails) plus two

[270] http://www.junit.org (2001-02-16).

`Objects`. Actually, there is even more overloading, and there are eight different resolvents to `assertEquals`:

```
assertEquals( Object expected, Object actual ) -> boolean
assertEquals( vector expected, vector actual ) -> boolean
assertEquals( number expected, number actual ) -> boolean
assertEquals( number expected, number actual, number delta ) ->
    boolean
 + four functions with additional message strings

assertBagEquals( bag expected, bag actual ) -> boolean
 + with additional message string
```

The reason for overloading with `number` arguments is that this makes it possible to compare an `integer` with a `real` (i.e. type checking is overridden). A test which I belive would be useful for scientific and technical database implementations is the one which takes *three* `number` arguments. The third argument `delta` is the maximum allowed difference between the `expected` and `actual` numbers for the assertion to be valid. In this way rounding errors and approximations can be handled in assertions.

Two vectors pass `assertEquals` only if they are equal element by element. This is handled fully by the Amos II core. The overloaded function simply is a means to introduce additional information upon failures. Note however, that the vectors {4,5} and{4.0,5.0} are *not* equal since the elements have different data types.

Two bags should be considered equal if the bags contain the same elements, irrespective of order. This is not handled gracefully by the Amos II system, which treats bag(1,2,3) and bag(3,2,1) as unequal. Thus, bag equality was implemented by means of bag differences:

```
if ( some( bagDifference(actual,expected) ) or
     some( bagDifference(expected,actual) ) )
then failEquals(message, expected, actual);
```

Equality tests on bags should use the function `assertBagEquals`. When I implemented this under the name `assertEquals` the Amos II system was not able to resolve the overloaded functions correctly.

Appendix E : The `FitModel` Strategy Pattern

As described in section 8.8, a `FitModel` object has both state (the values of its attributes `name`, `description`, and `formula`) and behaviour (methods to create the correct type and number of `FitParameter` objects). This behaviour is different for the different `FitModel` subtypes. The same applies for `WeightingModel`, where the behaviour which differs between subtypes is the way a vector of weights is calculated.

In this Appendix will be shown how the types `ExperimentEvaluation`, `Calculation-Method`, and `FitModel` (or `WeightingModel`) constitute a ”Strategy” design pattern[271]. In an object-oriented programming language such as Java, this design pattern consists of an abstract class[272] (”AbstractStrategy”) with one concrete subclass for each implemented algorithm[273], see Figure E1.

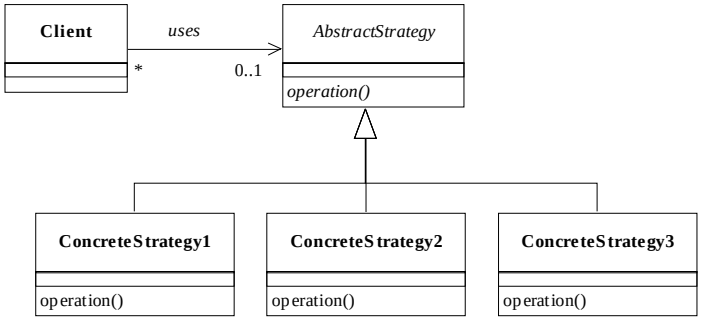


Figure E1. The Strategy design pattern as a UML class diagram. (After Grand 1998.)

This is similar to how `FitModel` (and `WeightingModel`) are implemented, although Amos II does not allow the designer to define abstract types (or to enforce singletons):

```
/* "Abstract type" FitModel */
create type FitModel properties (
   name charstring key, description charstring, formula charstring);

/* "abstract" method */
create function createFitParameters(FitModel,bag ligands,bag sites)
    -> FitParameter;

/* "Concrete subtype" Bindfit of FitModel */
create type Bindfit under FitModel;

/* "concrete" method */
create function createFitParameters(Bindfit, bag ligands, bag sites)
   -> FitParameter as
    begin
    . . . lengthy implementation . . .
    end;

/* Other "concrete" subtypes of FitModel */
    . . .
```

[271] Gamma, Helm, Johnson, Vlissides 1995; Grand 1998.

[272] An abstract class is a class without extent, i.e. no objects can be created of it. Abstract classes may include both concrete and abstract methods (with and without implementation, respectively). Concrete subclasses of an abstract class must implement all abstract methods of the superclass.

[273] The concrete subclasses are singletons, i.e. there is only one object of each class.

```
/* One instance of each "concrete" subtype is created,
   none of "abstract" supertype FitModel */
create Bindfit instances :bindfit;
set name(:bindfit) = "Bindfit";
set description(:bindfit) =
   "Reversible binding of N ligands to M sites";
set formula(:bindfit)= "B_i=\sum_{b=1}^{m}\left(\frac{K_{ib}F_iR_b}"
                     + "{1+\sum_{a=1}^{n}K_{ab}F_a}\right)+N_iF_i";
. . .
```

Figure E2 shows a UML class diagram of the Amos II types `FitModel` (with subtypes), `CalculationModel`, and `ExperimentEvaluation`. The diagram is drawn according to an object-oriented data model, e.g., as if `model` were an attribute of `CalculationMethod` (instead of a function which maps a `CalculationMethod` object onto a `FitModel` object). The type `CalculationMethod` takes the role of `Client` in Figure E1, and `CalculationMethod` is in turn called by `ExperimentEvaluation`.

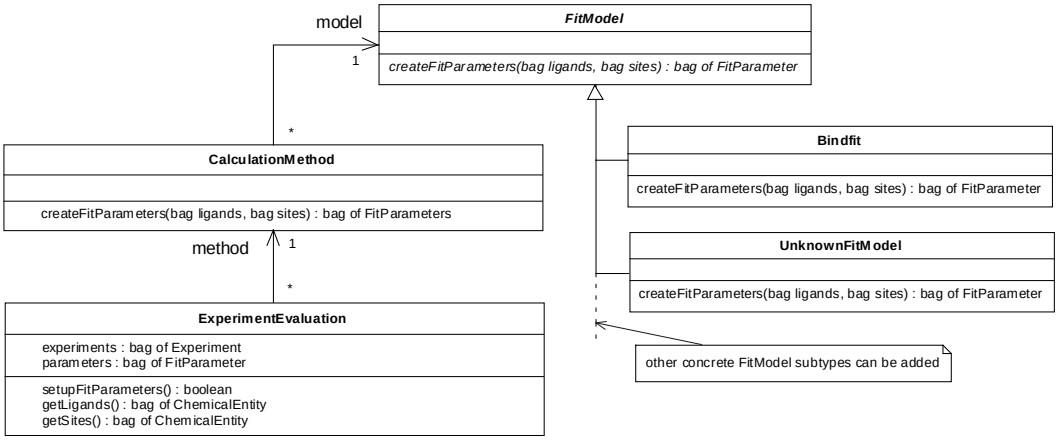
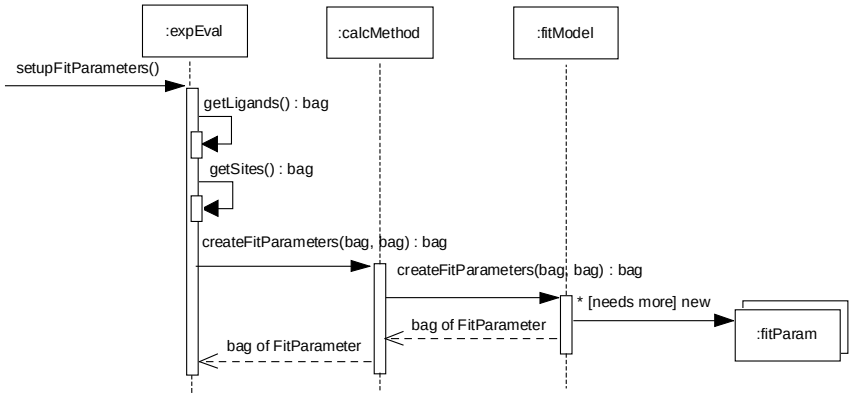


Figure E2. UML class diagram of the `FitModel` Strategy pattern.

Similar diagrams could be drawn for `WeightingModel`, whereby the `NoWeighting` subtype of `WeightingModel` would corresponds to a ”Null object” pattern[274].

The UML sequence diagram of Figure E3 shows how the appropriate set of `FitParameter` objects are created for an `ExperimentEvaluation` object (cf. section 8.8.1).



[274] Woolf 1997; Grand 1998.

Figure E3. UML sequence diagram for the creation of fit parameter objects.

Appendix F : Polymorphic Behaviour by Applying Functions

As described in section 8.8.3, the use of the foreign function `apply` is an alternative to subtyping through inheritance when implementing the ”Strategy” pattern. The foreign function is implemented in Lisp, and is presently not included in the Amos II core. It cannot be used in the official Amos II version, but only in the ”developers version” (camos2), where it is possible to go out to Lisp and define new functions. The Amos II foreign function `apply` does in turn use the Lisp function *apply*.

The foreign function `apply` was implemented by Tore Risch (Dept of Information Science, Uppsala University) who is principal investigator in the Amos II database engine research project, as well as supervisor for this Thesis. The impetus for this implementation was discussions between Tore and me as to whether AMOSQL allows a user to invoke a function he/she has a reference to. This was not possible, and the security aspects of including this feature in Amos II have not yet been fully investigated.

F.1 How to Use the `apply` Function

The interface of `apply` is `apply(function f, vector parameters) -> vector results`, i.e. all arguments to the function f should be packed into a vector, and the results are returned in a vector. The following code excerpt shows how different weighting models are implemented as instances of type `WeightingModel`, with an attribute `getWeightsFunctionName`. The function `getWeights` takes the `WeightingModel` object as argument, and gets the appropriate function with `functionnamed`. This function is then ”applied”, and the contents of the result is returned as a vector of real weights[275]. (A foreign function is used since Amos II can not handle exponentiation.)

```
create type WeightingModel properties ( name charstring key, . . . ,
    getWeightsFunctionName charstring );

create WeightingModel (name, . . . , getWeightsFunctionName)
    instances
    :noweights ("No weighting", . . . ,
                                    "getUnityWeights"),
    :munsonrodbard ("Munson & Rodbard weighting", . . . ,
                                    "getMunsonRodbardWeights"),
    :delean ("DeLean weighting", . . . ,
                                    "getDeLeanWeights");

create function getDeLeanWeights(   vector values,
                                    vector weightingParameters )
    -> vector of real
    as foreign "JAVA:Prot1Weighting/deleanWeights";


create function getWeights( WeightingModel wm, vector values,
                            vector weightingParameters )
```

[275] Note that ”apply(---)” will give us a vector of results. I.e., since our result is a vector (of weights) this vector will be the single element of another vector ({ { 0.01, 0.012, 0.014,…} }). To extract the elements of a collection we use `in`. A bug in Amos II prevents us from using direct addressing to get the first element of the result vector. If there had been more that one argument, we could have wrapped the method containing ”apply(---)”.

```
    -> vector of real
    as select in( apply( functionnamed( getWeightsFunctionName(wm) ),
                         {values, weightingParameters} ) );
```

F.2 A Comparison With Java

Usually, we apply a function by explicitly giving its name in an AMOSQL statement, e.g. `name(:donald);` applies (or invokes) the function `name` on an object `:donald` (presumably of type `Person`). Similarly, in Java we invoke the method `getName()` by `donald.getName();` (where, presumably, the variable `donald` is assigned to an object of class `Person`).

The possibility to use the functions `functionnamed` and `apply` together in AMOSQL does at least partially corresponds to one of to Java's reflection capabilities (see the Java reflection API (*java.lang.reflect Class Method* 2001)).

We saw the implementation in AMOSQL in the previous section, an example of use is:

```
/* object :delean assumed setup appropriately */
set :values = {10.0, 11.0, 12.0, 23.0, 29.0, 45.0};
set :weightingparameters = {1.0e-6, 1.0e-3, 1.5};
set :weights = getWeights( :delean, :values, :weightingparameters );
/* now :weights = {31.6218,27.4094,24.0557,9.06576,6.40325,3.31268} */
```

The corresponding pattern implemented in Java:

```
/* object wm assumed assumed setup appropriately */
double[] values = {9.0, 11.0, 12.0, 23.0, 29.0, 45.0};
double[] weightingparameters = {1.0, 1.0, 1.0};
double[] weights;
weights = wm.getWeights( values, weightingparameters );

class WeightingModel {
   . . .
   public double[] getWeights( double[] values, double[] wp )
   {
      double[] result = null;
      Class c = WeightingModel.class;
      Class[] parameterTypes =
         new Class[] { double[].class, double[].class };
      Method weightingMethod;
      Object[] arguments = new Object[] { values, wp };
      try
      {
         Method weightingMethod = c.getMethod(
            getWeightsFunctionName, parameterTypes );
         result = (double[]) weightingMethod.invoke( null, arguments );
      } catch (NoSuchMethodException e) {
           System.out.println(e);
      } catch (IllegalAccessException e) {
           System.out.println(e);
      } catch (InvocationTargetException e) {
           System.out.println(e);
      }
      return result;
   }
} // class WeightingModel
```

There are two major differences: (i) In Java we would expect the class `WeightingModel` to implement all weighting methods, and also have an attribute `getWeightsFunctionName` discriminating between them. In the Amos II/Lisp solution, the type `WeightingModel` only has the discriminating attribute. The functions to calculate weighting methods are implemented independent of any user-defined type. Similarly, the Java method `getMethod` is bound to a specific class, while the Amos II `functionnamed` is not. (ii) With the Java reflection API the method `getMethod` takes a vector of parameter types as argument to help resolving overloaded methods. In the Amos II/Lisp solution this resolution is made dynamically and transparently, with help of the types in the vector of arguments to `apply`.

It seems easier to use the ”apply function” approach in Amos II than to use `Method.invoke` in Java. However, the comparison is not altogether fair since the Java reflection API contains a lot more features than invoking a method.

Appendix G : Integration of External Data

One of the main purposes of the PAQS project is to create a means for information integration by defining a common data model in the mediator and wrapping external data sources. Such wrappers have not been implemented in the prototype developed for this Thesis. However, this Appendix will demonstrate the use of external data sources with a small example. Furthermore, the Appendix will give examples of information integration by means of *derived types* (G.3) and *integration union types* (G.4).

The examples are also available from the author (fbn@hig.se) in electronic format as four files ("ChemicalData.mdb", "Register_ODBC_data_source.txt", "External_demo.amosql", and "External_IUT_demo.amosql").

G.1 The Relational Database

As an example of a relational data source we take a very small MS Access database of chemical data, ”ChemicalData.mdb”. We may access the data in the ChemicalData database through an ODBC[276] wrapper in the Amos II system[277]. Note that the database of chemical data we will use is for demonstration purposes only, it has fewer columns than a commercial molecular properties database would have, and there are *very* few rows in each of the tables.

Although the database is small, it is representative of how data would be stored in a larger commercial relational database of chemical properties. The relational database schema for ”ChemicalData.mdb” is the following:

```
ORGANIC_COMPOUNDS(
    Substance, Formula, MolecularWeight, Density, DipoleMoment,
    HeatOfCombustion )
INORGANIC_COMPOUNDS(
    Substance, Formula, MolecularWeight, Density, DipoleMoment,
    Solubility, SolubilityCode )
ELEMENTS(
    Znuc, Element, Symbol, AtomicWeight, Density, DominantIsotope,
    PercentageDI, RadioactiveIsotope1, PercentageRI1, Radioactive-
    Isotope2, PercentageRI2, RadioactiveIsotope3, PercentageRI3 )
```

The meaning of the various attributes should be clear from their names, and a more detailed description is given in the MS Access database. Underlined attributes have been declared primary keys. (These keys, and the unique fields `Element` and `Symbol` in `Elements` are indexed in the MS Access database. This is not necessary for the Amos II access.)

[276] ODBC (Open Database Connectivity) is a standard interface designed to provide interoperability between relational products.

[277] The Amos II system wrapper for ODBC data sources is described in (Brandani 1998).

G.2 Access the External Data Source from Amos II

It is assumed that the external MS Access database has been registered as an ODBC data source under the Windows operating system, and also within the Amos II system (see supplementary material, "Register_ODBC_data_source.txt").

After having registered the external database under the name 'ds' we may investigate the data source by retrieving metadata, access the data source by executing SQL commands through the ODBC API, or access the data source through the ODBC wrapper. Examples are given in the following sections.

G.2.1 Query Metadata

```
tables('ds');
columns('ds', 'ELEMENTS');
primary_keys('ds', 'ELEMENTS');
```

G.2.2 Execute SQL Commands through the ODBC API

One way to access the database is by direct SQL statements[278] through the ODBC API:

```
sql('ds','select znuc, element from elements order by znuc', {}, -1);
set :sql_string = 'select element, dominantisotope, percentageDI, ' +
                  'radioactiveisotope1, percentageRI1 from elements ' +
                  'where percentageDI < ? and percentageRI1 > ?';
set :parameters = {50.0, 10.0};
sql('ds', :sql_string, :parameters, -1);
```

If the external database was *not* defined as a read-only ODBC data source insertions, updates and deletions are possible too.

However, this direct use of the ODBC API is not a suitable means for data integration. For example, we use AMOSQL to access local data, but SQL2 syntax to access the external data. There is no location transparency, and the impedance mismatch remains.

G.2.3 Import Tables Through the ODBC Wrapper

The second way to access the external database is through the ODBC wrapper. When a table is imported into Amos II the relational table becomes represented by a proxy type of the Amos II functional data model. This proxy type can then be queried almost as a stored type[279]:

```
import_table('ds', 'ELEMENTS');
```

[278] The syntax is `sql(odbc_ds data_source, charstring sql_string, vector parameters, integer max_rows) -> bag of vector results`. I.e., the first argument 'ds' is the name we have registered the external database under, and the last argument is the max number of rows we want the ODBC connection to return. (We know there cannot be more than a little bit over 100 rows in the `Elements` table, but there might be thousands of small polar molecules stored in the table `Organic_Compounds`.) Sql strings containing question marks are filled in with the elements of the vector of parameters.

[279] The data in the external database can be *queried*, but we cannot *insert* or *update* data through the proxy type.

```
import_table('ds', 'Inorganic_Compounds');
import_table('ds', 'Organic_Compounds');

select znuc(e), element(e) from elements@ds e;

select oc into :nitmet
    from Organic_Compounds@ds oc
    where substance(oc) = "nitromethane";
dipole_moment(:nitmet);
```

It is easy to mix stored types and proxy types in the AMOSQL query, e.g. in the following join of `Organic_Compounds` and `ChemicalEntity`[280]:

```
select substance(oc), dipole_moment(oc)
    from Organic_Compounds@ds oc, ChemicalEntity ce
    where substance(oc) = name(ce);
```

G.3 Data Integration by Derived Types

In the previous section we saw how external data could be incorporated by first importing tables and then querying these tables in AMOSQL. However, Amos II has more powerful data integration primitives: object-oriented views implemented as derived types and integration union types (Josifovski and Risch 1999).

A *derived type* (DT) is a subtype of one or several existing (stored or proxy) types while an *integration union type* (IUT) is a supertype of several existing types. An example of using derived types is given in this section, and one with IUTs in the following.

We continue the example from above and create a derived type (DT) `Organics` as a subtype of the user-defined stored type `ChemicalEntity` and the proxy type `Organic_compounds@ds` which represents an external data source. The extent of `Organics` is restricted to those instances which satisfy the `where` clause in the following definition:

```
create derived type Organics
   subtype of Organic_compounds@ds oc, ChemicalEntity ce
   where name(ce) = substance(oc);
```

We may now query the new type `Organics` transparently, just as an ordinary user-defined Amos II data type. The following query example shows that we can use one local function (`name(ChemicalEntity)`) and one attribute of the relational table (`dipole_moment`) in the same query transparently:

```
select name(o),dipole_moment(o) from organics o;
```

For a fuller description of the functions available for the Amos II `odbc_ds` data type, see the Amos II User's Manual (Flodin et al 2000).

---

[280] In this and the following examples it is assumed that the scripts "prot1_chemicalentity.amosql" and "prot1_chemicalentityExamples.amosql" have been run. Alternatively, the following definitions and instantiations can been made:

```
create type ChemicalEntity properties ( name charstring key );
create ChemicalEntity (name) instances
   :methanol ("methanol"), :ethanol ("ethanol"), :methane ("methane");
create ChemicalEntity (name) instances
   :iodide ("I-"),:triiodide ("(I3)-"),:iodine ("I2");
```

G.4 Data Integration by Integration Union Types

An *integration union type* (IUT) is a supertype of several existing types. Suppose, for example, that we wish to import the two external tables `Organic_Compounds` and `Inorganic_Compounds`, and then make a union `Compounds` of them. We can then use the latter type, `Compounds`, transparently as an Amos II type[281]. This should be fairly simple since we expect no semantic heterogeneity between them (see, however, footnote 282). It might also be useful to integrate this type `Compound` with the local type `ChemicalEntity`, but then quite a lot of reconciliation between the schemas has to be made.

The use of integration union types (IUTs) in Amos II is technically a bit more complicated than the use of derived types. At the present status of Amos II IUTs are mainly intended for integrating data from two or more external data sources. The scenario described by Risch and Josifovski (2001) is that data from two relational databases should be integrated. Each database is being wrapped by an Amos II mediator and a third mediator imports the types of the two mediators, and integrates them, possibly together with a locally stored type, as an IUT.

The demonstration example given below integrates data from two relational tables of the same database as one IUT. Thus, only two mediators are used:

- "Mediator_ChemData" wraps the ODBC data source (a "translator", cf. Figure 10). This mediator imports the two tables of organic and inorganic substances from the MS Access database. When it is running, it listens for clients. The mediator also functions as nameserver.
- "Integrator" creates an IUT from two proxy types which represent the two wrapped tables. The "Integrator" first registers with the nameserver, and then imports types from the mediator of the ChemData database:

```
import_db('Mediator_ChemData');
import_type('Organic_Compounds_ds', 'Mediator_ChemData');
import_type('Inorganic_Compounds_ds', 'Mediator_ChemData');
```

We can now create the integration union type. (Only a subset of the columns in the external database tables were used for this demonstration example.)

```
create derived type Compounds
  key charstring substance
  supertype of
    Organic_Compounds_ds@mediator_ChemData oc    = substance(oc),
    Inorganic_Compounds_ds@mediator_ChemData ioc = substance(ioc)
  functions
    ( name charstring, u_formula charstring,
      u_solubility real, u_dipole_moment real )
    case oc:
      name = substance(oc);
      u_formula = formula(oc);
      u_solubility = 0.0;
      u_dipole_moment = dipole_moment(oc);
    case ioc:
      name = substance(ioc);
      u_formula = formula(ioc);
      u_solubility = solubility(ioc);
      u_dipole_moment = dipole_moment(ioc);
    case oc, ioc:
```

[281] Since IUTs and DTs are based on Amos II proxy type it is presently not possible to update the data source.

```
        name = substance(ioc);
        u_formula = formula(ioc);
        u_solubility = solubility(ioc);
        u_dipole_moment = dipole_moment(ioc);
    end functions
    properties (CAS_nr charstring);
```

Now we can query the two MS Access tables as one single Amos II type.

```
select name(c), u_formula(c), u_dipole_moment(c)
   from Compounds c where u_dipole_moment(c) > 10.0;
```

In the definition of the IUT we first declare two proxy types to use, then list a few functions the IUT should have, and then give implementations for the functions. (In the present case the two tables are expected to be non-overlapping, so `"case oc, ioc"` will likely never occur[282].) Finally, there is a possibility to define new attributes to an IUT, i.e. attributes existing in neither of the data sources (`CAS_nr` above).

Note that I have chosen to set the solubility to zero for organic compounds in the example. Since the table does not list water solubilities for organic compounds an alternative would be to set all such attributes to `null`. However, then the inorganic and organic compounds would be treated unequally. The data source for inorganic compounds actually has a lot of `null` values in the solubility column, but these are set to 0.0 when imported through the ODBC wrapper[283]. Hence, if a missing solubility in the inorganic compounds data source results in 0.0 for the IUT, then so should all missing organic compound solubilities.

It is worth noting that I have not managed to create an IUT from one proxy type and one (local) stored type. This seems not possible in the present Amos II version. I.e., it is not possible to create an IUT of `ChemicalEntity` and `Organic_Compounds_ds@mediator_ChemData`. In order to do that we need to have `ChemicalEntity` in *another* mediator. Then, we could import two types, and construct an IUT of them. Neither is it possible to create an IUT from a stored type and a proxy type for an imported table (i.e. importing the table directly, and not going via a second mediator). It is possible, however, to create an integration union type from two *stored* types. This is probably of little practical use, since it is easier to use the inheritance primitives of the type system.

[282] Actually, in the tables that were used to populate the database (Aylward and Findley 1994) a few substances exist as inorganic as well as organic compounds (e.g. CCl_4). Interestingly, the data are heterogeneous in two ways: (1) Some molecular properties are given with a higher precision in the table of organic compounds, and (2) the thermo-chemical data refer to the gas phase and to the standard state at 25 °C for organic and inorganic compounds, respectively. In the terminology of section 4.3.3, case (1) is a "data conflict" (the two data sources have different values), and case (2) is a "semantic conflict" (different meaning of the same concept). Thus, even the task of integrating data from two tables *of the same data source* could be quite demanding.

[283] When executing an SQL command through the ODBC API, we have the possibility to exclude null values: `sql2('ds','select substance, formula, solubility from inorganic_compounds where solubility <> null',{},10);` A corresponding possibility does not seem to exist when going through the ODBC wrapper.