# Concept Instance Sketching and Design for a Biological Database Framework

Alan McCulloch[1], Pieter Demmers[2], Jason Mitchell[1], David Townley[3], Paul Smale[1], Russell Smithies[1], Craig Miskell[1], Anar Khan[1], Nauman Maqbool[1]

[1] AgResearch NZ Ltd [2] Crop and Food Research [3] CSIRO/Sheep Genomics

## Abstract

We have developed a biological resource description framework (BRDF) based on a collection of concept instance sketches expressed using an extended hypergraph notation, and have applied this framework to the data warehousing requirements of a number of research projects in the fields of genomics, genetics, nutrigenomics and molecular biology. The resulting framework and its instances have been implemented in a Postgres database, with a Python object oriented API and web-based interface, and a number of production systems have been built and operated using the framework, and continue to be developed. This paper defines and describes the framework and concept instance sketching technique we have developed, and briefly describes the implementation of the systems based on this framework and our experience with them to date.

## 1. Introduction

The main focus of this work was applied rather than theoretical – the requirement was to develop data warehouse solutions to support a number of multidisciplinary research programs. These programs were expected to generate diverse and dynamic datasets in both human and agricultural research projects, including gene expression work (such as microarray experiments running on various platforms e.g. clone based arrays, commercial oligo, and Affymetrix; quantitative and real time PCR; EST sampling and clustering), proteomics results, metabolite measurement experiments, genotyping experiments, genomic sequencing, rich phenotype data, epidemiological case control diet studies and high throughput food fraction functional assay experiments.

The data warehouse solutions were intended to be long term repositories with a requirement for flexible support for both internal and external ontologies, free text comments attached to any database object; warehousing of associated binary files such as images and protocol documents; and fine grained record access control to partition warehouse stakeholders as may be required. As well as supporting these immediate projects, the warehouses were intended to develop and demonstrate capability to support future projects. While the primary focus was on data warehousing and data mining rather than data management, we expected the repositories would be required to assist with some data management tasks associated with data cleaning and checking - for example, resolving conflicting genotypes; and hosting both raw and normalised expression data, and array spot images, for quality control review.

In view of the highly multidisciplinary nature of these programs, we made the decision to develop a single data framework, rather than federate a number of existing domain specific databases as was initially considered – for example a federation of a microarray repository, sequence repository etc. A federation would only address some of the project threads, leaving us still with a significant residual database development

requirement. We also expected considerable operational problems federating databases each living within separate secure intranets.

## 2. Results

The requirement was met by developing an extensible data framework. We developed an extended hypergraph based concept instance sketching technique to solve the difficult problem of bridging the gap between a large and poorly defined data modelling problem, and the tightly specified entity relationship models typically required by the software engineering design-validation and build process. The rest of the paper will:

- define what we mean by a data framework, and why conceptual and data modelling languages cannot completely specify a data framework
- describe what we mean by concept instance sketching, how this helps in specification of a data framework and why we believe our approach is novel
- present the novel extended hypergraph concept instance sketching technique we developed
- explain the need to use hypergraphs rather than graphs for concept instance sketching
- provide examples of the concept instance sketches developed to date
- describe the implemented framework and systems resulting from this work and our experience with them to date
- suggest future work.

## 3. Definition of a Data Framework

In this paper a "data framework" is defined as:

1. a database schema, that we know in advance will be extended and modified over the project lifetime, together with

2. a set of rules for deciding which changes to the schema are permitted according to the framework, and how they are to be implemented. These rules encompass, for example, table naming conventions, and a strict classification systems for tables, so that each addition to the schema must be classifiable into one of the categories supported by the framework

Thus a data framework in this context is an extensible database schema – but with definite limits on the types of extension permitted and how they are implemented.

## 4. Modelling Languages Cannot Completely Specify a Data Framework

Methodologies such as Entity Relationship (ER) modelling are used to formally and completely specify data models, but cannot completely specify data frameworks as we have defined them – since data frameworks include statements outside the model itself, about how the model is permitted to be modified.

In order to make statements about how a data schema is permitted to change - i.e. to specify the data framework - we need to be able to make statements about the types of entities and relationships that are supported by the framework, so that we can specify what sorts of extensions to the schema are permissible within the framework. This in turn requires us to elucidate and survey the types of data relationships that are expected to be conserved across the revisions and extensions that will be experienced over the project lifetime.

# 5. Concept Instance Sketching

We define concept instance sketching as the graphical depiction of *instances* of data objects and the relationships between them. Unlike conceptual modelling which has an extensive literature and well developed formalisms, concept sketching of instances of data relations as a design technique does not appear to have been much discussed. (Ehresmann introduced a mathematical structure called a sketch[1] that provides an abstract generalisation of graphs and graphical languages. Note that our use of the term sketch is not directly related to this theory).

For example we may sketch an instance of a regulatory relationship between biological sequences, for the sequence part of a bioinformatic warehouse:

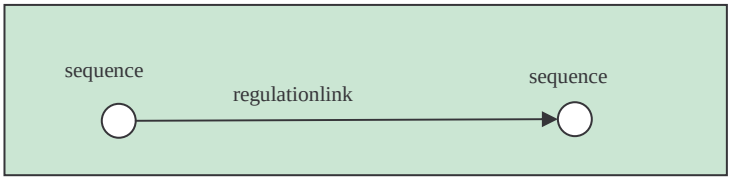


**Figure 1**

Concept instance sketches, which depict instances of relationships, should be explicitly distinguished from concept models, since the respective diagrams are superficially and misleadingly similar. The above diagram could not be a concept or data model, since the sequence entity appears more than once, whereas entities may only appear once in data model diagrams. Concept instance sketches convey the structure of the data model "by example" using instances of objects and relations, rather than by making statements about the abstract object and relation types in a graphical modelling language.

We have used concept instance sketching to elucidate in turn a patchwork of overlapping patterns of data relationships, each involving usually no more than four or five types of object. The above relationship is an example of the Link table relationship class, in our BRDF framework. Other examples appear below.

We have found that a series of concept instance sketches of small overlapping parts of a large and complex data model is a very useful part of the initial design process, and in the design of subsequent extensions to the model. Concept instance sketches depict actual instances of data relations, rather than the more abstract entities and their relationships and so are more intuitive, and more directly mirror the actual structure in the data. The restriction of each sketch to a small part of the overall model renders the complex whole comprehensible as a series of simple parts each of which may be grasped easily.

# 6. Inadequacy of Graphs for Concept Instance Sketching – Hypergraphs.

Graph based techniques such as ER diagrams and petri nets are actually formal modelling (visual) languages whereas our concept instance sketches are pictures that are isomorphic to relation instances, intended to show "by example" the structure of the data - they are not at all linguistic devices. Using a graph consisting of pairs of vertices connected by edges, we are restricted to isomorphic pictures of at most binary relationship instances. (Techniques using graphs to implement visual modelling languages are not so restricted, as the pictures are formal statements in a modelling language rather than pictures of data instances, and these languages are adequately expressive). A hypergraph is a graph in which an "edge" may connect more than two vertices:

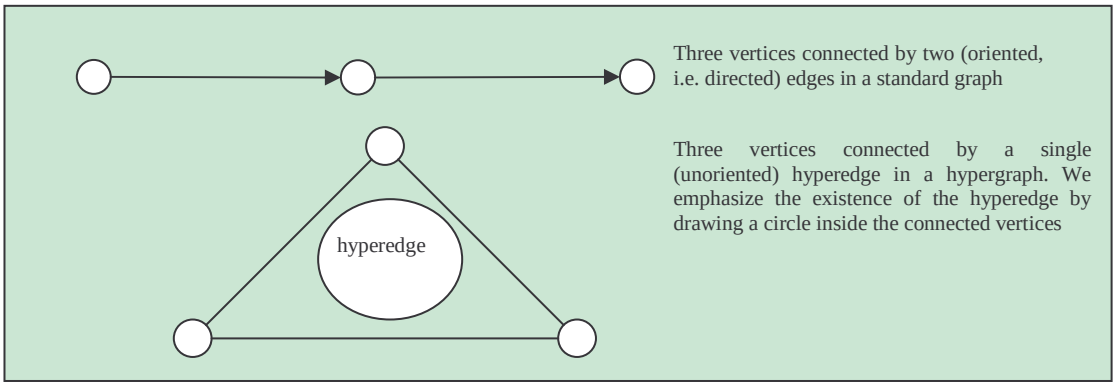


**Figure 2**

Using hypergraphs we are able to sketch instances of higher order relations. There are previous reports of the use of hypergraphs to depict data instances[2] although not in the context of concept instance sketching as part of the design process. Our sketches extend the hypergraph structure to allow dangling edges connected to only a single vertex

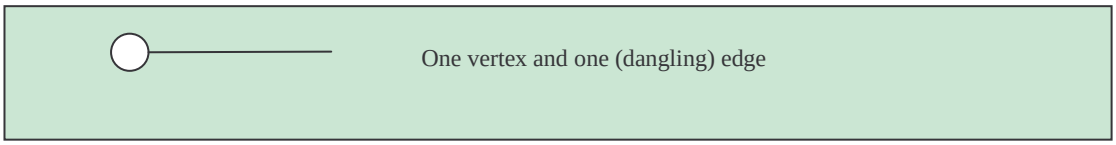


**Figure 3**

We refer to these structures that include dangling edges as extended hypergraphs. The reason for this extension is to enable us to depict instances of the "star schema" pattern often used in data warehouses.[3] This is a design in which a number of "fact" tables are related to a given entity, each recording some distinct data dimension associated with the entity. We handle these as unary relations, hence depicted by a dangling edge.

A further point to note about our sketches is that a hyperedge may also appear in another context as a vertex. This is because most relation instances in the database are also objects in their own right and can appear as terms in some other relationship.

We developed around 25 concept instance sketches of parts of the model. Some examples are included in the text below.

# 7. Data Framework Specification

The results of our concept instance sketching were used to help classify the types of table supported by the framework – these sketches reveal common patterns across different data relations, that enable us to classify these relations and the corresponding database tables. Any new tables added to the schema must be assignable to one of these classes, and table naming is determined by the class of table. These framework rules help control and limit the complexity of the schema as it extends.

All tables are classed as either “ob” or “op” tables. Ob tables represent objects or entities and correspond to the vertices in our extended hypergraph concept sketches. Op tables represent relations between obs and correspond to the hyperedges in the sketches.

An Op may connect from 1 to many obs (an op “connecting” only one ob is diagrammed as a dangling edge in the concept sketches). The abbreviation "op" has been used rather than "edge" because it is shorter. It is motivated by the fact that each edge can also be thought of as an operation involving the vertices that it connects.

All ob and op instances in the system are assigned a unique numeric id, so that both the vertices and hyperedges are named uniquely.

## *7.1. Op Tables*

Ops correspond to hyperedges (including dangling edges) in the concept instance sketches, and represent relations between obs. Dangling edges are unary relations and represent fact dimensions as in data warehouse star schema designs. Most ops inherit from the ob table so that an instance of a relation may itself be involved in a relation. This is done because we often want to be able to record information about instances of relations – usually by attaching one or more fact tables to the corresponding table. There are five standard types of op tables in the BRDF framework – Fact, Study, Observation, Link and Function tables.

### 7.1.1. Fact Tables

A fact table stores a specific dimension of information about a single database object, and does not involve any formal relationship to any other database object. It is sketched as a dangling edge, connecting to a single object.

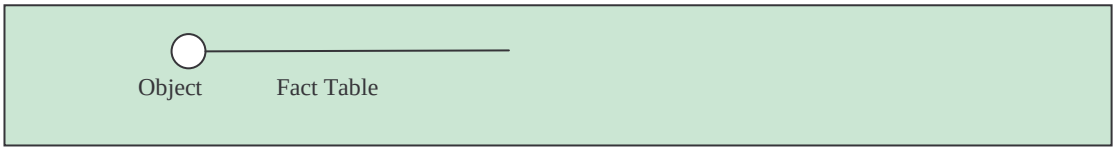


**Figure 4**

There may be many such fact tables connected to a given object in a star pattern, each dealing with a different data dimension.

### 7.1.2. Link Tables

A Link Table stores details of binary many-to-many relationships between two database objects. For example, the geneticOblistMembershipLink relation allows a gene to appear in a number of different lists, and a list to contain many genes. The geneticNomenclatureLink relation associates one or more ontologies or ontology terms with a gene – for example each gene usually has an ontology set up specifically to list all the synonymous gene symbols; and genes are associated with Gene Ontology[4] terms.

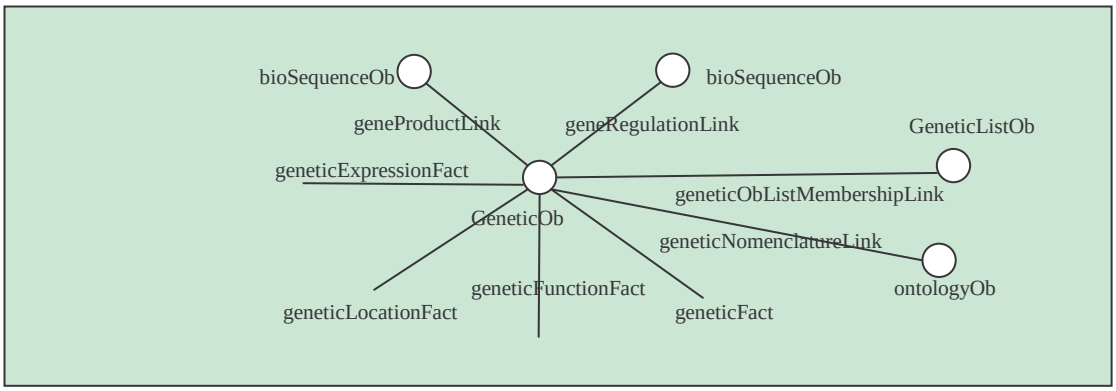


**Figure 5 Fact and Link Tables**

Each distinct hyperedge, including dangling edges, is realised as a table in the schema. Thus the above diagram would be realised as 12 database tables. (Note that the repeated biosequenceob instances only require one biosequence table)

### 7.1.3. Function Tables

Function tables store instances of ternary or higher relationships. For example the sequencingFunction table stores attributes of a sequencing run, specifically attributes of a relation involving a sequence, a sample and a lab Resource list (vectors, primers etc). Attributes of this relation would include who, when and possibly some run parameters.

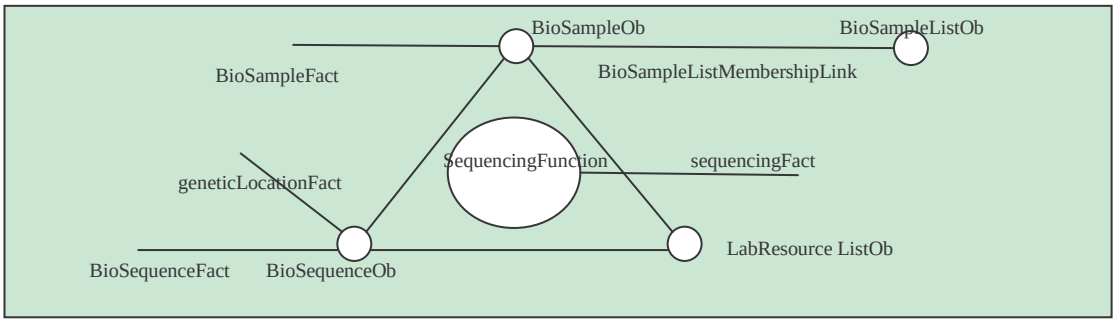


**Figure 6 Function Tables**

# 8. Study and Observation Tables

A “study” groups together a collection of observations of a certain type, together with metadata about the collection. An observation is always an observation of something – hence an observation is conceptually sketched as a link between a master study object, and the thing being observed. For example, in a microarray study the thing being observed is a microarray spot, while in a qPCR expression study the thing being

observed is a particular primer probe. In a genetic study involving genotyping, the thing being observed is a particular genetic test e.g. a particular SNP.

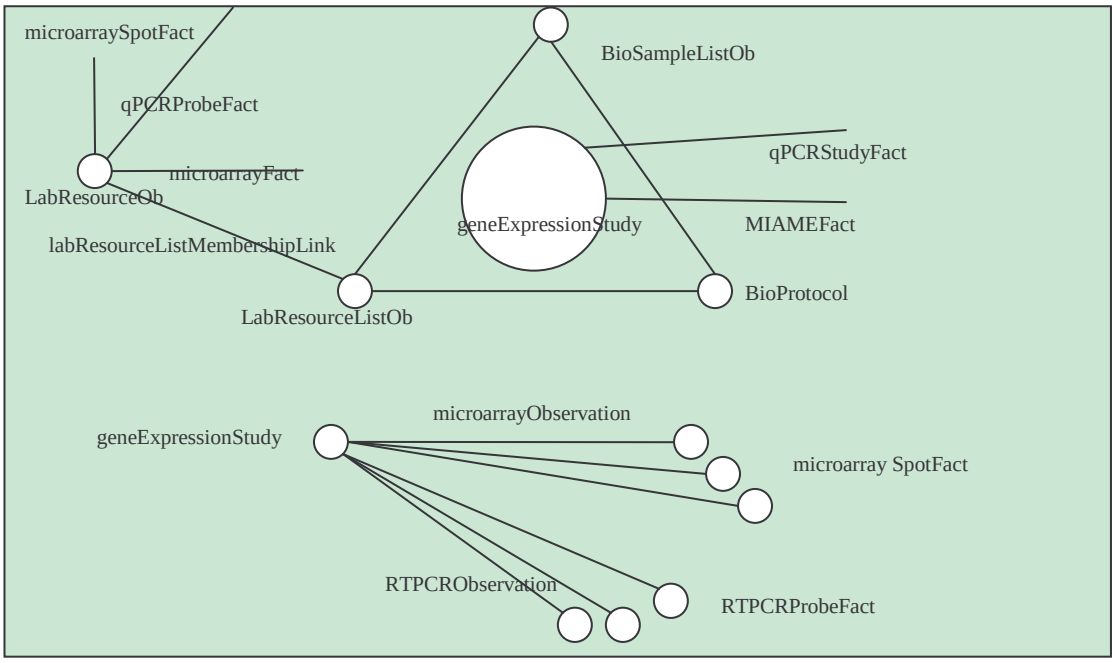


**Figure 7 Study and Observation Tables**

A gene expression study instance is sketched as a hyperedge connecting one or more samples used in the experiment, the list of lab resources used, and a protocol. In the lower half of the diagram, gene expression studies appear in another context, as a vertex connected via observation links with microarray spots – an expression observation is a relation between a study and a spot. The gene expression study hyperedge is also in another guise a vertex in a unary relation with two fact tables which record qPCR and MIAME[5] fact dimensions.

The BRDF framework models other types of experiment, including *in silico* "experiments" such as BLAST searches, using the same study, observation and fact table design patterns.

# 9. Extending the BRDF Schema

Any table added to the BRDF framework must be classed as either

1. an ob table and represent a new type of entity in the schema
2. one of the five types of op table – fact, link, study, observation or function

These restrictions are partly enforced by the use of an inheritance mechanism in both the database schema and Python object oriented API.

# 10. Implementation

We proceeded straight from the concept instance sketches to realisation in a Postgres object relational database, and then reverse-engineered a formal ER model using the diagramming tool Visio in order to check details and consistency of the data model. Each distinct vertex, and each hyperedge and dangling edge, of the concept sketches are

realised as distinct tables in the schema. The BRDF schema currently consists of some 85 tables, classified as follows:

| | |
|---|---|
| Fact tables | 29 |
| Link tables | 16 |
| Study tables | 5 |
| Observation tables | 6 |
| Function tables | 5 |
| Ob tables | 20 |
| Framework | 4 |

"Framework" tables refer to a small set of tables that are used to record the hypergraph structure of the schema – i.e. which tables correspond to hyperedges or dangling edges, and which tables are linked to others via hyperedges. Although this structural information could probably be deduced from the Postgres data dictionary itself, we found it convenient to record this directly in our own data dictionary layer. We have used this data dictionary layer to support an auto-generated web based user interface, allowing users to navigate the schema, so that they can (for example) click from a given record, to all other records reachable from that record via any hyperedge that links the two. This provides a fairly rich set of automatic reports without any additional coding – for example all spots on a microarray; all experiments that have used a given microarray chip; all genetic tests recorded for a subject; all subjects that have been surveyed with a given genetic test ; all BLAST searches that have been done against a given database; etc.

We have also developed an object oriented Python API for the framework. Each class implements basic services for a database object, such as how to display itself as HTML, how to initialise itself from the database, and how to commit an update to the database, where we make the object editable. Related classes and methods are collected together into a Python module – there are currently 22 modules.

## 11. Experience To Date

We are currently running five production instances of this framework for five distinct science programs. Datasets that have been warehoused to date include microarray experiments on a number of platforms including clone based two-colour arrays based on ruminant EST clones, commercial mouse and human two-colour oligo arrays, Affymetrix system ruminant and plant arrays; various sequencing projects, including ESTs and genomic sequencing, providing feature visualisation, built-in trace file viewing; a set of food compounds and various types of functional assay, part of a nutrigenomics research program, together with human genotyping and rich phenotype dataset from the same program; a ruminant gene index; *in silico* results – BLAST runs and other types of database searches.

Microarray data has mostly been imported from standard data files in formats such as Genepix GPR and Affymetrix CEL. Many datasets have been bulk imported from existing spreadsheets since the data was generated prior to the warehouse, or spreadsheet based lab data management was already entrenched. Processing a multitude of spreadsheets in various highly customised formats from a number of different labs is time consuming, and we have found it important to work with scientists and technicians

to adapt their spreadsheets so that data is maintained in a simple tabular format so that it can be easily warehoused. All of our data import procedures are coded in such a way that they can be used to handle interactive data entry from web pages, as well as batch imports, so that we could over time move from spreadsheet based data handling, to a direct web based system. We expect this to be a slow process since local spreadsheets are such a convenient and efficient tool for data entry in the lab environment.

The framework has enabled us to quickly roll out warehouses to a number of projects, and extend the schema for individual projects as required, while maintaining control of complexity. Developers with no previous Python or Postgres experience have been able to become productive quite quickly in maintaining and developing the framework API. We have used several of Postgres' object relational features – user defined stored functions are especially powerful. Particularly with rich and dynamic phenotype datasets, it is generally necessary to store these in a simple (essentially non-relational) table of key-value pairs. When combined with stored functions to access and return named phenotype traits, this approach works extremely well and is very cost-effective.

We have been able to quickly develop reports for extracting raw and normalised expression data, and develop graphical displays of gene expression across multiple experiments, and simple web-interfaces to review spot thumbnail images from multiple experiments, as part of quality control to check correct dye assignments, and to scrutinise over and under saturation of images at various scan intensities.

Reports have also been developed to integrate genotpye and phenotype information and these have proven invaluable in resolving genotype lab and data processing errors and achieving a coherent and integrated genetic analysis.

A powerful keyword search engine using PL/pgSQL has been developed, which enables us to search across all data types and tune the balance between search specificity and sensitivity.

# 12. Future Theoretical and Practical work

## *12.1. Theoretical Work*

We have developed a simple concept instance sketching technique which we have found very useful in solving a difficult data modelling problem. However we acknowledge that this technique is incomplete and informal. Because our technique is not a modelling language, the opportunity for formalism is probably limited. Nevertheless one line of future work would be to define the technique more formally than we have done. Furthermore we have already encountered areas where the technique needs extending – for example some vertices of a hyperedge may be optional, but we have not attempted to develop any diagramming convention to indicate this.

We proceeded straight from our concept instance sketches to realisation in a database schema, then reverse engineered the schema to obtain an ER diagram when we needed to check aspects of the model. It should be possible to solve for the model directly from the concept sketches. Clearly there would be in general no unique solution, however there would be an optimal solution. An algorithm for finding an optimal model

consistent with a set of concept instance sketches would make the sketching technique even more useful and put it on a firmer footing.

# 13. Practical Work

### 13.1.1. User Interfaces and Data Mining

One of our main focuses for future development is in development of the database browser user interface, mainly in the direction of providing single integrated views of data and images; and in developing additional data extract reports and extending current reports. While the default "auto-generated" user interface allows a user to navigate to any record in the database by following links (i.e. traversing hyperedges), and this works very well for a database administrator, this is not convenient for scientists, for whom a single view that integrates linked data into a single page is much more useful. We are actively working to improve the user interface based on feedback and suggestions from users.

### 13.1.2. Data Warehousing – warehouse rationale and challenges

We believe it is important when setting up warehouses of this nature that both developers and stakeholders maintain ongoing clarity about the rationale for the existence of the warehouse, and what services and benefits it should provide over and above public repositories, and freely or commercially available tools. There is little point in reinventing the services provided by superb public sequence and expression repositories (such as offered by NCBI), and in view of the very rapid development of these and other public repositories and tools, careful attention must be paid to development priorities. The most distinctive service provided by warehouses of the type we have set up is probably the integration of sequencing, expression, phenotype and other experimental results generated as part of a coordinated science program. This means that while improving user interfaces and developing mining tools is important, equal priority must be given to ensuring that the various datasets that we hope to integrate are actually captured and curated adequately. This last task is a challenging one, since it may take some time for this activity to bear visible fruit, and the generators of the various datasets may find difficulty according the same priority to long term warehousing as do the custodians of the warehouse.

Acknowledgements : This research was supported by SheepGenomics which is an initiative of Australian Wool Innovation Limited and Meat & Livestock Australia together with 11 leading research organisations in Australia and New Zealand. This is SheepGenomics publication number 85. We also acknowledge the support of Nutrigenomics New Zealand

References

1. C. Wells. Sketches: Outline with references (1994). On ftp //ftp.cwru.edu/math/wells.*.
2. Watters, C., Shepherd, A.A., A transient hypergraph-based model for data access, ACM Transactions on Information Systems Volume 8, Issue 2 (April 1990)
3. Utley, C. Designing the Star Schema Database
4. http://www.geneontology.org/
5. Brazma, A. et al Minimum information about a microarray experiment (MIAME)—toward standards for microarray data Nature Genetics, vol 29, no. 4, pp 365-371