Homogenising access to heterogeneous biomedical data sources

Erwin Bonsma and Jeroen Vrijnsen

Philips Research, High Tech Campus 37, 5656 AE Eindhoven, The Netherlands

Abstract This paper reports our experiences of developing data access services in the context of the ACGT project. The paper documents two aspects of the work that we carried out. First the focus is on the problem of how to best provide a syntactically homogeneous data access interface for a set of heterogeneous data sources. We describe related work, outline the approach we have taken, and report our findings. The second part of this paper documents integration issues that we encountered when realizing the data access services. Choices with regards to realization have significant impact on the time and effort that is needed to develop and maintain the services and our experiences may provide useful guidance to others wanting to develop similar functionality.

Introduction

The work reported here has been carried out in the context of the ACGT (Advancing Clinico-genomic Trials on Cancer) project. The aim of ACGT is to develop open-source, semantic and grid-based technologies in support of post-genomic clinical trials in cancer research [1]. One of the main challenges in carrying out post-genomic research is to efficiently manage and retrieve all relevant data. Carrying out a post-genomic clinical trial involves the collection and storage of a wide variety of data, including: clinical data collected on Case Report Forms (e.g. symptoms, histology, administered treatment, treatment response), imaging data (e.g. X-Ray, CT, MR, Ultrasound), and genomic data (e.g. microarray data). Next to that there are many public biomedical databases that are relevant. These store information about gene and protein sequences, pathways, genomic variation, microarray experiments, medical literature, tumour antigens, protein domains, metabolites, etc. Biomedical researchers currently have to use many different tools and web interfaces to find and extract the data that is relevant to their clinical research. Providing seamless and integrated access to clinical, genetic and image databases would therefore greatly facilitate post-genomic research.

In order to provide seamless access to a heterogeneous set of databases syntactic and semantic integration needs to take place. Syntactic data integration handles differences in the formats and mechanisms of data access, whereas semantic integration deals with the meaning of information; it must handle the fact that information can be represented in different ways, using different terms and identifiers.

With regards to syntactic heterogeneities, the main areas where databases differ are:

- access protocols, e.g. SOAP/HTTP, DICOM, JDBC,
- data formats, e.g. different formatting of date values,
- message formats, e.g. XML, HTML, protocol-specific, and
- query mechanisms, e.g. SQL, literal matching, keyword-based search, or protocol-specific.

An example of a query mechanism specific to the biomedical domain is BLAST [2], which is used by sequence databases. Matching is approximate and parameters can be specified controlling the accuracy and speed of matching. A completely different query mechanism is needed to access medical image data, which is standardised using the DICOM protocol [3]. DICOM does not allow complex queries, as it does not intend to provide a generalized database query mechanism [4]. The baseline query functionality is very basic, and the optional extended query functionality is still limited and eccentric.

Semantic integration in ACGT is handled using Query Translation, carried out by a semantic mediator that uses a Local as View approach. It accepts queries expressed in the ACGT Master Ontology, divides them in sub-queries, and translates each to the ontology used by the underlying database. The remainder of this paper focusses on the syntactic integration of data sources. For details about the semantic integration approach, please refer to [5].

Related work

Syntactically homogeneous access to distributed data sources is typically provided by way of wrappers [6, 7, 8, 9]. One of the main challenges in building wrappers is the variation in the query functionality of the underlying data sources [10]. Data sources may not only use different data models and syntactically different query mechanisms, but their query capabilities can differ as well. This makes it difficult to support a common query language, an essential step towards syntactic homogeneity. There are two extreme approaches [7]. A highly expressive common query language can be chosen. This, however, makes it difficult to implement wrappers for sources with primitive query capabilities. Furthermore, if the wrappers are used by a mediator, it means that query decomposition, subquery scheduling and result composition may be done by both; the mediator must be able to decompose queries across multiple data sources and a wrapper for a data source must be able to decompose a complex query into simpler ones that the data source can handle. This means duplication of implementation effort but also leads to overall sub-optimal query execution performance. On the other hand, if a very basic common query language is chosen, significant and unnecessary performance penalties are introduced as the capabilities of the underlying data sources are not effectively used.

As neither approach is ideal, an intermediate solution is proposed in [7]. A powerful common query language is chosen, but wrappers may choose to only support a subset of the queries, based on the capabilities of the underlying data source. Each wrapper describes the queries it supports using the Relational Query Description Language (RQDL) developed for this purpose. An RQDL specification consists of a set of query templates that represent parameterized queries that are supported. RQDL uses a context-free grammar to describe arbitrarily large sets of templates. Templates can be schema-independent as well as schema-dependent. Benefits of this approach are that wrappers can provide and expose query functionality that better corresponds to that of the underlying data source. A drawback is the increased complexity associated with interpreting and reasoning about the query capabilities of each source, but feasibility is demonstrated by the Capabilities-Based Rewriter, described in the same paper, that uses the wrappers and produces query execution plans in reasonable time.

A more recent example, applied in practice to life sciences data, is given by DiscoveryLink [8], a database middleware system for extracting data from multiple sources in response to a single query. The system consists of two parts: a wrapper architecture, and a query optimizer. SQL is used as the common query language for the wrappers, but wrappers may only support a subset of SQL. In the simplest case, a wrapper retrieves a projection over all rows in a given table. Wrappers can, however, also indicate that they support filtering conditions, or joins, and if so, how many. The paper proposes to involve wrappers in the query optimization process. Wrappers are asked for estimates on the query execution time and expected size of the result set for different sub queries. The query optimizer will use this information when deciding how to decompose the query. It requires efficient communication between the query optimizer and the wrapper, which is made possible because wrappers are shared-libraries, co-located with the query optimizer.

EDUTELLA [11] uses an RDF query language that has various language levels, with increasing functionality. The basic level supports RDF graph matching, the level above that adds disjunction, and the use of recursion in queries is added at even higher levels. Support for aggregation is an optional feature, orthogonal to these levels. Wrappers can support the level of the query language that best fits the query capabilities of their data source.

It is generally recognized that writing wrappers requires significant programming effort, and as a result significant research efforts have been devoted to automating parts of this (see e.g. [6], [9]). In general, automation is focused on a subset of the different data sources, e.g. sources with a web interface [12].

Approach

We identified the following functional requirements for the data access services. Firstly, they should provide a uniform data access interface. This includes uniformity of transport protocol, message syntax, query language, and data format. Secondly, they should export the structure of the database, using a common data model, together with possible query limitations of the data source. Clients of the web service require this information for constructing queries. Thirdly, they should enforce the data source access policy, and audit access to data sources. For post-genomic clinical trial data, there exist strict legal and ethical requirements that need to be adhered to.

A common query language is needed to achieve a uniform interface. It needs to meet various requirements. Firstly, it must be sufficiently expressive; it should support the types of queries that clinicians and biomedical researchers want to carry out. Secondly, it must be attainable, with acceptable effort, to map the query language to those used by the various data sources that need to be accessed. Thirdly, it must be convenient to use the query language for semantic mediation, the next step of the data integration process. Fourthly, it should be a community accepted standard. This ensures that there are sufficient support tools available, such as parsing and query engines, and also increases the possibilities for our approach to be eventually widely adopted. We have chosen SPARQL [13] as the query language, as it satisfies all these requirements.

Web Services have been chosen as the common interface technology within ACGT, as this technology suits the distributed nature of the project with respect to the data, computing resources, and development teams. For the data access services we decided additionally to use OGSA-DAI, a Web Services framework for data access [14]. It uses an activity framework that enables flexible service invocation, and re-use of common data access functionality. The results of queries will be returned using the SPARQL Query Results XML Format [15], which is the natural choice given the web services context and the use of SPARQL.

To meet the second requirement each data access service exports its schema using RDF Schema [16]. This is the standard way to describe RDF data sources, which is how the data sources appear given that SPARQL is used.

Access to each data source is controlled by integrating the data access service into the ACGT security infrastructure. Authentication is credential-based and delegation of credentials between services is supported. Authorization is controlled centrally and authorization decisions are, amongst others, based on membership to virtual organizations, which can be created as required.

Implementation

We have implemented data access services for three data source types: relational databases, medical image databases, and microarray databases. These databases have been chosen after careful review of requirements; they are considered the most important in the context of post-genomic clinical trials given the data-mining scenarios that were identified during the requirements-gathering process.

Figure 1 shows the data access services in the context of the data analysis architecture. The workflow enactor carries out data-mining workflows. It uses the semantic mediator for retrieving data. The latter accepts queries expressed in the ACGT Master Ontology, and converts them to the local ontology of the data source that is queried. The query results are converted in the opposite direction. Before a data access service handles a query, it checks whether or not the user is authorized to access the data source by contacting the authorization server. The data access services handle SPARQL queries from the semantic mediator. Additionally, they may also be contacted directly by the workflow enactor. This is the case for retrieval of image and assay files, which do not require semantic mediation. The requested data is typically not returned to the workflow enactor, but delivered to file at a specified temporary storage location. The workflow enactor receives the unique identifiers for files that have been created, which it can forward to the data-mining service so that the latter can retrieve and analyse the data.

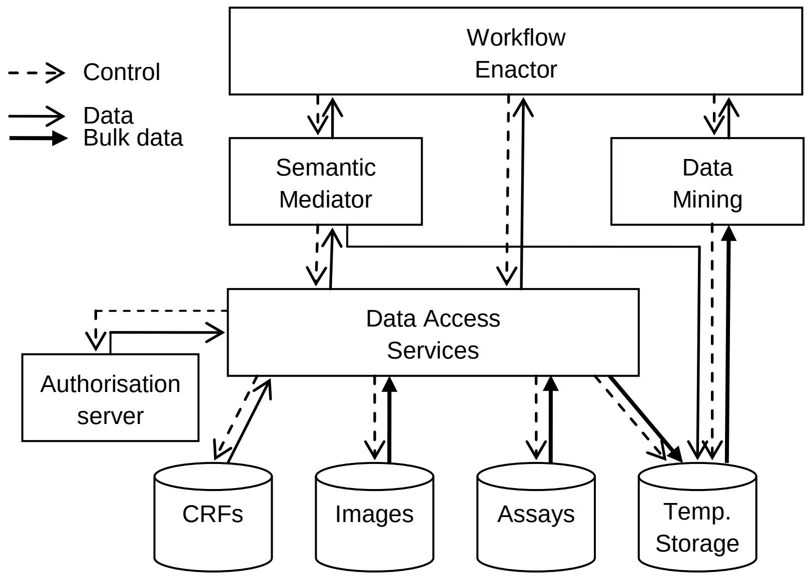

Fig. 1. The data analysis architecture of ACGT

There are two relevant aspects with regards to the terminology we use. First of all, we use the term “data access service” to refer to a class of services, e.g. the DICOM data access service, as well as for referring to specific instances, e.g. the data access service for DICOM database X. The distinction should always be apparent from the context. Secondly, each data access service is not actually a standalone web service. Within the OGSA-DAI framework multiple data access services are deployed as different data resources within a single OGSA-DAI web service. This has implications for the addressing of the data access services, but is not important for the remainder of this paper.

Query functionality

For the implementation of the query functionality for relational databases it is necessary to translate queries from SPARQL to SQL. For this, we are using the Open Source package D2RQ [17]. It can wrap a relational database into a virtual, read-only Jena RDF graph [18], rewrite SPARQL queries and Jena API calls into application-datamodel-specific SQL queries, and transform the returned data ino RDF triples. We therefore only had to integrate this functionality into the OGSA-DAI activity framework.

Realizing a data access service for medical image databases requires more effort. First of all, custom code is needed to implement the query translation. As the DICOM information model maps naturally to RDF, it is relatively straightforward to express DICOM queries in SPARQL. However, the DICOM standard only provides limited query functionality, which means that only a subset of syntactically valid SPARQL queries can be expressed as DICOM queries. For the initial implementation, we only support SPARQL queries that can either be directly converted to a DICOM query, or that can be handled using a single DICOM query combined with filters at the data access service that do not require temporary storage of query results (i.e. any query match that is returned by a DICOM server is either immediately discarded, or after optional conversion, immediately returned to the client). This way, the data access service does not need to store intermediate results, and implementation is significantly simplified. Figure 2 shows an example of a supported SPARQL query for a DICOM image repository.

For the medical image data access service, image retrieval functionality was also added; the ability to query the image metadata is of limited use if the actual images cannot be retrieved. The retrieval functionality has been implemented using OGSA-DAI’s activity framework so that it can be invoked in various ways. For example, a single request message can be used to query the image metadata, and to asynchronously retrieve and deliver the corresponding images.

```
PREFIX dicom: <http://example.philips.com/dicom/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?name ?dob ?studyId ?studyDescr
WHERE {
  ?patient dicom:PatientsName ?name ;
           dicom:PatientsBirthDate ?dob .
  ?study dicom:Patient ?patient ;
         dicom:StudyID ?studyId .
  OPTIONAL {
    ?study dicom:StudyDescription ?studyDescr .
  }
  FILTER ( ?dob >= "1970-01-01"^^xsd:date &&
           ?dob <  "1980-01-01"^^xsd:date )
}
```

Fig. 2. Example of DICOM query expressed using SPARQL.

Our third data access service provides access to the BASE database, a database for storing the results of microarray analysis [19]. The data access service interacts with the BASE database by way of a Web Service interface. The current implementation of the data access service provides retrieval of assay files, given their unique identifiers. More advanced query functionality is not provided, as this has not been needed yet. Typically assay files are obtained by first querying the clinical data, e.g. for all patients with an ER-negative tumor that responded positively to treatment, and next retrieving the corresponding assay files from BASE.

Miscellaneous functionality

Due to the heterogeneity of the data sources, each data access service requires code that is specific to its type of data source. However, the different data access services also need to provide common functionality, which offers the opportunity for code reuse. The main mechanism by which the OGSA-DAI platform encourages the reuse of code is through its activity framework [14]. Requests from clients to an OGSA-DAI data resource can contain multiple activities, linked together into a pipeline. For example, the first activity may comprise a query to a DICOM server for a set of image identifiers. A second activity may extract the identifiers and retrieve the corresponding images. The images may be fed to a third activity, which compresses the image data, and feeds the resulting archive file to a fourth activity, which delivers the archive to a specified FTP server. Although the interface of the query activities for the different data access services is typically the same (thus providing a homogeneous interface), their implementation is typically highly dependent on the type of data source that is queried. Activities further in the pipeline are typically more generic and their implementation may be reused by multiple data access services. The OGSA-DAI platform comes with a large set of generic activities, but we developed additional ones for use by our data access services. One example is an activity for delivering files to the Gridge Data Management System [20], which we use for temporary storage, using myProxy certificates for authentication. Another activity can calculate checksums for data-streams. It can be used for testing service functionality after changes to the implementation, as well as for carrying out periodic liveness tests of a running service. We also extended the default ZIP activity so that it can pack multiple files into a single archive.

Integration experiences

The realization of the data access services requires integration of a large number of third-party software libraries. There are two reasons why a large number of third-party packages is needed: firstly, the complexity of the software stack associated with (grid-based) Web Services, the interface standard chosen in ACGT, and secondly, the heterogeneity of the underlying databases, which typically each have their own sets of standards and APIs associated with them. The software stack for the data access services consists of the following layers:

- data access services
- OGSA-DAI
- Globus
- Tomcat

The lowest layer is the Tomcat web service container, which hosts the web services. Globus sits on top of Tomcat; it is used for implementing the certificate-based security framework. The layer above that consists of OGSA-DAI. It provides a modular, activity-based data access framework for use by the layer above it. The top layer consists of the data access services which handle query and result transformation, and data retrieval and storage for the supported data sources. Each class of data access service depends on various third-party libraries for its implementation. For example, the relational data access service uses D2RQ [17] to translate SPARQL queries to SQL, which in turn uses Jena [18]. The DICOM data access service uses Jena as well, together with dcm4che [21] for accessing DICOM servers. The BASE data access service uses client-code provided by the BASE developers for accessing their BASE web service.

Given this setup, one of the biggest problems is managing the depencies between all different third-party software packages that are used. This is especially challenging because all data service resources are deployed within the same OGSA-DAI instantiation and third-party packages (deployed as Java jar files) that are needed by only one or a few of the data access services are visible to all. This can lead to dependency conflicts between data access services that are otherwise independent. A pair of data access services that can individually be deployed successfully inside an OGSA-DAI instantiation may not necessarily be successfully deployed alongside each other.

Three concrete issues that we encountered may help to illustrate the types of integration problems this gives. Firstly, after we had discovered and reported a bug in a third party library we used (Jena), we could not deploy the release that included the fix, as this new release was incompatible with another third party library (D2RQ) that we were using

Secondly, we have experienced problems deploying compiled and packaged code provided by other partners in ACGT, which was due to a slight incompatibility in an underlying third-party library (Axis) provided by the version of the (Globus-based) web service container that was used. Fortunately, the incompatibility did not exist at the source code level, so rebuilding the code with the third-party libraries of the container where the service was to be deployed fixed the problem.

We encountered a third problem after we had upgraded the web services container, which was required to fix a depency conflict. One of our services would now hang when handling requests. As it turned out, this was due to a change of the third-party library implementing the JavaMail API, which resided three layers below our code. It was due to a more strict implementation of the JavaMail API, which in turn revealed a bug in another third-party library (Axiom), which relied on a more lenient interpretation of the API's contract in order to function correctly.

It is worth pointing out that in all three cases, the fact that source code was available for all third-party software components greatly helped in tracking down and solving the problem.

Discussion

We have implemented OGSA-DAI data access services for three types of data sources: relational databases, medical image databases and a micro-array database. The main research question is how to best provide a syntactically homogeneous interface, and a key question is the query language that is used. We have chosen SPARQL as the common query language and have demonstrated that it can be successfully applied to relational databases and DICOM image databases.

For the relational databases, the SPARQL language does not support all features offered by the query language of the data source, SQL. For instance, it does not support aggregation of data (averaging, summation, counting, etc). So aggregation needs to be performed at the client-side, even though the underlying database supports it directly, which negatively affects performance. The actual use of the system by the end users will clarify whether this is a problem that needs to be addressed.

For medical image databases, SPARQL is more expressive than the query support provided by the DICOM protocol. For this reason, the data access service does not support all queries. These limitations are currently described as text, but should be expressed in a more formal manner, so that other services and applications can interpret these and handle accordingly. In order to select a suitable formal framework for this, we need to thoroughly review the capabilities and limitations of all relevant data sources.

A capability-restricted data access architecture has the advantage that it is easier to develop data access services for data sources; as a consequence, new data sources can be integrated much more quickly. It may, however, complicate applications and services that use the data access services. A higher level data access service may therefore be introduced that hides query restrictions of the underlying services. This generic service would decompose queries for a specific data access service as need be, store the intermediate results, and join these to produce the final answer. This would facilitate implementation of the semantic mediator, while incurring a slight performance penalty. However, this higher-level data access service may also carry out generic optimizations such as caching of query results, resulting in performance gains.

Another open issue is how to provide text-based query functionality. There are many public biomedical databases where part of the data is free text. Examples are descriptions of microarray experiments (e.g. in GEO [22] and ArrayExpress [23]), descriptions of gene and protein functions (e.g. in UniProt [24] and EntrezGene [25]), and abstracts and titles of medical publications (e.g. in PubMed [26]). Although most databases provide keyword-based functionality for querying data, this method of searching is not directly supported by SPARQL, so it is not immediately obvious how to extend the current data access services interface to support this functionality. One approach would be to add a separate text-based query interface for data sources that support this. This exposes more details of the underlying data source, resulting in a less homogeneous interface. This is undesirable but may be unavoidable in practice. However, there is a more important question that needs to be answered first: how should querying of text data be handled by the semantic layer? This is an important question as it determines the query interface that is available to end-users, but answering it falls outside the scope of this paper.

To give an impression of the overhead caused by the use of data access services, compared to direct interaction with the databases, we can report the results of performance experiments that we have carried out. The amount of overhead depends on various factors, including the complexity of the query, the amount of results that are returned, and the underlying database. For simple queries the performance degration could be as much as a factor hundred (in particular for the relational database, which responds very quickly). For more complex queries the overhead decreased significantly, down to a factor of two (for the DICOM database). Overhead was similarly low for retrieval of bulk image and microarray data, but high for retrieval of bulk data that is returned in the XML response message. The latter is due to limitations of the API for constructing the response message, which needs to be constructed entirely in memory before it can be sent to the client.

Finally, many of the problems encountered when deploying data access services for heterogeneous data sources are of a practical nature. For reasons of scalability, all data access services are deployed in the same web services container. This implies however that they run inside the same virtual machine, which can lead to unexpected conflicts. The complexity of the grid-based web services stack in combination with the need to use many third-party libraries, each with their own dependencies and particular implementations of part of the web services stack, makes it a challenge to resolve dependency conflicts.

Acknowledgments This research has been supported by the Advanced Clinico-Genomic Trials in Cancer (ACGT) project (FP6 IST-2005-026996) funded by the European Commission. Thanks to the anonymous reviewer for the useful feedback.

References

[1] Tsiknakis M, Kafetzopoulos D, Potamias G, et al (2006) Building a European Biomedical Grid on Cancer: The ACGT Integrated Project. In: Studies in Health Technology and Informatics, Volume 120

[2] Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ (1990). "Basic local alignment search tool". J Mol Biol 215 (3): 403–410

[3] Bidgood Jr WD, Horii SC, Prior FW, van Syckle DE (1997) Understanding and Using DICOM, the Data Interchange Standard for Biomedical Imaging. J Am Med Inf Assoc 4(3).

[4] National Electrical Manufacturers Association (2004) Digital Imaging and Communications in Medicine (DICOM), Part 4: Service Class Specifications, Annex C: Query/Retrieve Service Class, 27-74

[5] Martín L, Bonsma E, Potamias G, et al (2007) Data Access and Management in ACGT: Tools to solve syntactic and semantic heterogeneities between clinical and image databases. In: LNCS 4802, 24-33.

[6] Papakonstantinou Y, Gupta A, Garcia-Molina H, Ullman J (1995) A query translation scheme for rapid implementation of wrappers, In: LNCS 1013, 319-344

[7] Papakonstantinou Y, Gupta A, Haas L (1998) Capabilities-based query rewriting in mediator systems, Distrib and Parallel Databases, 6:73-110

[8] Haas LM, Schwarz PM, Kodali P, et al (2001) DiscoveryLink: A system for integrated access to life sciences data sources. IBM Syst J, 20(2):489-511

[9] Thiran P, Hainaut JL, Houben GJ (2005) Database Wrappers Development: Towards Automatic Generation, In: Proc 9th Eur Conf Softw Maintenance and Reengineering, 207-216

[10] Hernandez T, Kambhampati S (2004) Integration of biological sources: Current systems and challenges. ACM SIGMOD Record 33 (3), 51-60

[11] Nejdl W, Wolf B, Qu C, et al (2002) EDUTELLA: A P2P networking infrastructure based on RDF. In: Proc 11th Int World Wide Web Conf (WWW2002)

[12] Liu L, Zhang J, Han W, et al (2005) XWRAPComposer: A Multi-page data extraction service for bio-computing applications. In: Proc 2005 IEEE Int Conf on Serv Comput, 271-278

[13] Prud'hommeaux E, Seaborne A (2007) SPARQL Query Language for RDF, W3C Candidate Recommendation, http://www.w3.org/TR/2007/CR-rdf-sparql-query-20070614/. Accessed 25 Feb 2009

[14] Antonioletti M, et al, The design and implementation of grid database services in OGSA-DAI. In: Concurrency and Computation: Practice and Experience, 17(2-4):357-376

[15] Becket D, Broekstra J (2008) SPARQL Query Results XML Format. W3C Recommendation. http://www.w3.org/TR/rdf-sparql-XMLres/. Accessed 30 Mar 2009

[16] Brickley D, Guha RV (2004) RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation. http http://www.w3.org/TR/rdf-schema/. Accessed 30 Mar 2009

[17] Bizer C, Seaborne A (2004) D2RQ – Treating non-RDF databases as virtual RDF graphs. In: Proc 3rd Int Semantic Web Conf (ISWC2004)

[18] Carroll JJ, Dickinson I, Dollin C, et al (2003) Jena: Implementing the semantic web recommendations. Technical Report HPL-2003

[19] Saal LH, Troein C, Vallon-Christersson J, et al (2002) BioArray Software Environment: A Platform for Comprehensive Management and Analysis of Microarray Data. Genome Biology 2002 3(8): software0003.1-0003.6

[20] Pukacki J, Nabrzyski J, et al (2006) Programming Grid Applications with Gridge, Comp Methods in Sci and Technol 12(1):47-68

[21] Open Source Clinical Image and Object Management, http://www.dcm4che.org/. Accessed 11 Feb 2009

[22] Gene Expression Omnibus (GEO), http://www.ncbi.nlm.nih.gov/geo/ Accessed 11 Feb 2009

[23] ArrayExpress, http://www.ebi.ac.uk/microarray-as/ae/. Accessed 11 Feb 2009

[24] Uniprot, http://www.uniprot.org/. Accessed 11 Feb 2009

[25] Entrez Gene, http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene. Accessed 11 Feb 2009

[26] PubMed, http://www.ncbi.nlm.nih.gov/pubmed/. Accessed 11 Feb 2009