

Design Studies Related to the Development of Distributed Web-based European Carbohydrate Databases

Design Study DS2: Creating a peer-to-peer network of distributed databases for applications related to Glycosciences.

Task Titles:

DS2-T2: First draft of a test implementation of P2P network and concept of central database.

Deliverable

DS2-D2: Report: "Implementation of P2P-network and central database".

Dissemination: PU; Partners: 1, 4

Due date of deliverable: 31.03.2007; Actual submission date: 31.03.2007

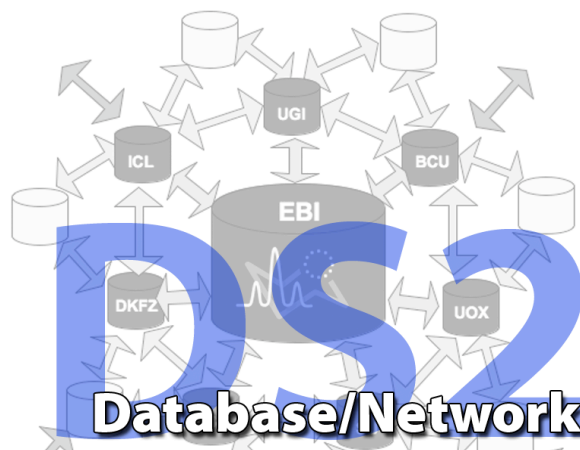
Start date of project: 01.04.2005; Duration: 48 months

Organisation name of lead contractor for this deliverable:

Deutsches Krebsforschungszentrum, Heidelberg, Germany

Authors

Hiren Joshi



Concepts for the development of the distributed EUROCarbDB

Hiren Joshi

April 20, 2007

Contents

1	Definitions	3
2	Introduction	4
2.1	Glycan databases	4
2.2	Centralised curation	5
2.3	Open curation	6
2.4	Peer to peer software	7
2.5	Goals of EUROCarbDB	7
3	System design	8
4	Network design	11
4.1	General network layout	11
4.2	Network layers	12
4.3	Data formats	12
4.4	Query layer	13
4.4.1	Protocols	13
4.5	Messages	14
4.5.1	Control messages	14
4.5.2	Query messages	16
4.5.3	Capability discovery	16
4.5.4	Message authentication	16
4.5.5	Action messages	18
4.6	Data layer	18
4.6.1	Protocols	18
4.6.2	Proteomecommons Tranche network	19
4.7	Data merging	21
4.7.1	The need for merging algorithms	21
4.7.2	Synchronisation process	22
4.7.3	Data canonicalisation and hashes	22
4.7.4	Data updates from core	22
4.7.5	Parallel insertion of duplicate entities	22
4.7.6	Update from local database to core	22
4.7.7	Publishing of records	26
4.8	Security	26
4.8.1	Public key cryptography	27
4.8.2	Digital signatures	27
4.8.3	Certificate mechanisms	27

5	Software implementation	28
5.0.4	Java and Tomcat	28
5.0.5	Hibernate	28
5.0.6	PostgreSQL	29
5.0.7	Struts 2	29
5.0.8	XML libraries	29
	References	30

1 Definitions

API Application Programming Interface

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

XML eXtensible Markup Language

MVC Model View Controller

ORM Object Relational Mapping

SQL Structured Query Language

CRUD Create, Read, Update and Destroy

OOP Object Oriented Programming

UniProt Universal Protein resource

NCBI National Center for Biotechnology Information

MeSH Medical Subject Headings

MEDLINE Medical Literature Analysis and Retrieval System Online

REST REpresentational State Transfer

XML-RPC XML Remote Procedure Call

WSDL Web Services Description Language

PKI Public Key Infrastructure

DFS Distributed File System

GPL GNU Public License

GID Global Identifier

LID Local Identifier

DBMS DataBase Management System

CCRC Complex Carbohydrate Research Center

CCSD Complex Carbohydrate Structural Database

Glossary

Glyco-CT Sequence format for glycans yielding an unambiguous representation of structure

Glycome The total set of glycan structures found in an organism

JDBC Java Database Connectivity Application Programming Interface (API)

Nonce Key string used as an identifier for a particular conversation. The name is derived from n once, a number (n) that is unique (generated only once).

Object oriented programming A programming paradigm using conceptual objects to build applications

SOAP Protocol for eXtensible Markup Language (XML) based web services

Tranche Distributed file system for scientific data

2 Introduction

2.1 Glycan databases

The development and maintenance of knowledge bases - which encapsulate the current body of knowledge for a particular discipline - are key to the practice of the discipline. In molecular biology, the use of sequence databases is a standard procedure, as evidenced by genomic and proteomic disciplines using genome data and Universal Protein resource (UniProt) [1] as reference databases. The free availability of reference data has spurred development in this area, opening up new avenues for research. In contrast to the genomic and proteomic databases, glycomics is missing an appropriate database to act as a reference resource.

Since the late 80s, there have been multiple efforts to collect information on glycans and make it available as a reference database. For the majority of these efforts, data entered into the database has been sourced from previously published data. The curation of data from literature is an expensive and time-consuming exercise, requiring at least two dedicated glycoscientist curators, database IT staff, and a senior glycoscientist to oversee the management and procedure followed. In addition, effective mechanisms to keep track of new glycan publications, and access to the journals themselves are required.

Of the curation-based databases - KEGG [2], Glycosciences.de [3], BCSDB [4] and Glycominds [5] based their databases upon the early Complex Carbohydrate Research Center (CCRC) Complex Carbohydrate Structural Database (CCSD) [6] - also known as CarbBank. Each database has taken the CarbBank data, transformed and cleaned the data somewhat, and then made either a subset, or all the data available via the internet. Each database then supplemented their data with

new entries from additional sources. However, as each of the databases took a different approach to their designs, interoperability between the different databases still does not exist.

In addition to information from published sources, a lot of both structural and ancillary data is being generated in the various glycomics labs. All this data is currently being stored in an ad-hoc format, be it in labbooks, stored in a proprietary format, or simply sketched on a piece of paper. The lack of proper meta-data and annotation on this data means that over time, this data will be lost, and any kind of data mining cannot be performed. There's a clear need to provide methodologies for capturing this meta-data, and then providing a pathway for eventual publication and distribution of this data.

Currently, no repository of data exists that can enumerate all the structures found in any particular glycome. However, there is significant interest in the area of glycomics, and a large amount of data is being collected in individual labs. Unfortunately, this data is not being captured in a uniform way, and will likely remain stored using various proprietary schemas for the foreseeable future.

By creating an end-user curated database, it will be possible to create a long lived database resource that maintains quality and reliability.

2.2 Centralised curation

Databases have traditionally been maintained as centralised resources. This was often necessary because the infrastructure surrounding the database was focused on a single point of data entry for the database, and the databases servers themselves were expensive pieces of equipment. The last 15 years have seen the phenomena of the proliferation of commodity software combining with the internet and open source software to result in much of the expense of hosting a database being mitigated.

Although equipment costs are limited through the use of commodity hardware and software, maintenance and curation costs remain high for fully centralised databases. As the majority of data that was collected by centralised glycan databases required the review of literature and manual curation of data, the expense of such efforts often became prohibitive. This was due to the need for skilled professionals to be involved in the curation process. Depending on the curation procedure, data may need to be double checked to ensure the quality of the data, effectively doubling the workload.

A centrally curated database yields significant advantages in terms of the quality of the data, as the full process is controlled by the central group of curators, and no inconsistent data should enter the database. Naturally, the curation process is unable to fix problems in the data stemming from the article itself. For example, a lot of experimental meta-data is not published due to the lack of space in journals. Although the data may be of reporting quality in the literature, it may not be detailed enough for bioinformatic analysis.

Although the results from centralised curation are good, the process is highly

dependent on funding, and once the funding for a centrally curated database runs out, the database tends to stagnate. This is particularly evident with CarbBank and GlycoSuiteDB [7]. As the establishment of a glycome is a long term project, the longevity of any database has to be ensured.

2.3 Open curation

In contrast to a centrally curated database, more open databases allow the unrestricted entry of data by interested parties. This requires no outlay for curation, but maintaining the quality of the data is difficult. As a result of their open nature, publicly curated databases need to have measures in place to protect from bad data being inserted, both inadvertently and maliciously. These measures should attempt to ensure both syntactic and semantic integrity of the data as it enters into the database.

Ensuring syntactic integrity for a database can be achieved through the use of various automatic checking algorithms. Automatic checking must be applied to each record that is inserted into the database, checking the data against ontologies and vocabularies. The use of common ontologies and vocabularies reduces redundancy in the database, as data would only be represented canonically.

Semantic data is significantly harder to automatically maintain in a database. Although a record may be syntactically correct, the data that it represents may not be semantically correct. Automated semantic checking is currently not possible, so the best approximation to this is for peer reviews of the data to occur, ensuring that the data is correct in this way.

For open curation to work people need to be encouraged to contribute corrections to data, and a study of the methods to achieve this is beyond the scope of this document. However, there are certain principles that are adhered to in this design study which can be used to encourage peer review.

Ownership By promoting ownership of data entered into a publicly curated database, people are more likely to maintain the data. Appropriate crediting of data to the original source will mean that this person would be more amenable to make corrections to their data since their name, and reputation is at stake.

Linus' Law Named after open source programmer Linus Torvalds, this law states that "given enough eyeballs, all bugs are shallow". In this context, this means that with a sufficient level of peer review of records, the number of errors found in the database will tend towards zero. In addition to identifying the errors in records, it is also important that the peer reviewers have a method to modify the data so that changes can be made without onerous procedure. This follows the Wiki model for peer-review of data.

Tooling It is critically important that the appropriate tooling to facilitate the previous two points exists. Appropriate tools and user interfaces abstract away the complexity of the data entry process, and present a simple workflow to end users. Similarly, the benefits for the end user need to be clear - it must be obvious what the benefits of annotating data are.

2.4 Peer to peer software

Although the process behind curation can be distributed, efforts should also be made to decentralise the actual software comprising the database. For this purpose, a peer to peer concept has been developed for EUROCarbDB. Basing the database at a single institution is prone to failure as in the event of the database being shut down for an extended period of time, the entire database will not be available for usage. By hosting the database in multiple institutions, the longevity of the database can be ensured through redundancy.

The control of data is a significant point of contention for many scientists, and the ability to control the method of publishing the data is an important feature. Local databases which publish to a wider network of databases are appropriate here as the data can be collected on site at the lab, and then distributed to the peers. This network structure can also allow for ad-hoc networks to be formed, sharing data amongst collaborating partners, and allowing queries to be run across all participating nodes in the network.

2.5 Goals of EUROCarbDB

There are four goals for the peer to peer database design - to be distributed, to provide access to experimental data, to ensure the longevity of the database, and to maintain quality in the database.

Distributed Databases should work installed in local institutions to allow for private and unpublished data to be entered into the database, with a view to export the local data to the general network in the long term.

Accessible Each node of the EUROCarbDB network should provide access to experimental data from the site that it is installed at. This should be easily published to the wider community without requiring onerous technical procedures to be followed by the host institution.

Longevity The database should be designed to continue functioning, even after large nodes are shut down. For example, nodes hosted at the DKFZ and the EBI should not be required for the database and network to function. Fully automatic network reorganisation is not required as the network would remain largely static and any changes in the network topology can be applied manually.

Quality Measures should be in place to maintain the quality of data in the database. These should work in concert to promote the syntactic and semantic integrity of the records in a distributed environment.

3 System design

The system can be generally categorised as comprising four key components - Users, Software, Databases and the Network. Figure 1 illustrates the relationships between the different components. The user interacts with the software either through a HyperText Markup Language (HTML) interface, programmatically via the web services, or through a Java application. Web services are offered using the SOAP protocol. Although the first prototype of the application is a web-based application, the design has been chosen to accommodate the use of desktop applications as potential methods to deliver software. The software component interacts with the Tranche [8] network, experimental data and the backing database. Inter-node communication occurs via Tranche or through the web services interface. The system architecture is a variation on the three tier design for web-based systems. The three tier system encourages Model View Controller (MVC) design pattern usage, making the source code more maintainable and manageable. The MVC pattern can be mapped onto this design, where the model is represented by the Object Relational Mapping (ORM) and database components, view by the HTML and web services components, and the controller by the actions component.

Web-based model The decision to implement the EUROCarbDB application as a web-based application was made in order to develop and distribute the application in as short a time as possible. For web-based applications the use of HTML is a standard, and ensures greatest compatibility with users all over the world. Access to web services is not so standardised however, and the choice of SOAP as the protocol for access to the data was made in an effort to maximise interoperability.

Execution model The actual scientific functionality of the system will be encapsulated into controllers known as actions. Actions will act upon the network, experimental data and the ORM layer. The actions themselves are loosely coupled to the presentation and data models, which allows the actions to be purely functional modular components. This compartmentalisation of functionality makes it easier for third parties to add functionality to the system, allowing the software to grow in a manageable way.

Object Relational Mappings As stored in a modern DataBase Management System (DBMS), data is represented as sets of interrelated entities. Access to the data is achieved through the use of a specialised query language known as Structured Query Language (SQL). SQL allows for access to the Create, Read, Update and Destroy (CRUD) operations on the entities. However, the utility of SQL is limited

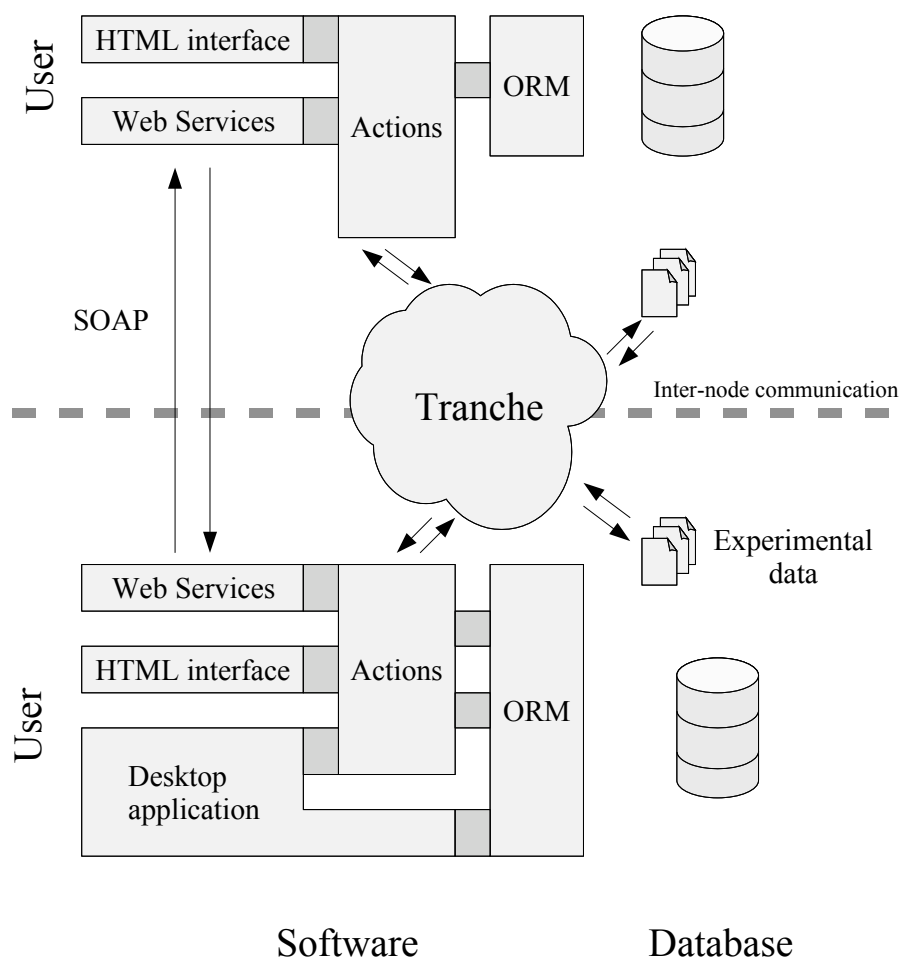


Figure 1: The overall architecture of two EUROCarbDB nodes communicating with each other. In the most simplistic representation of the system it is comprised of software and database components. Interactions between nodes occur via SOAP or through the Tranche network cloud intermediary. Experimental data associated with records on each node can be transferred via the Tranche network.

due to the scope of the language itself. For this reason, the application itself needs to be able to manipulate the data. Applications, generally speaking are written in an Object Oriented Programming (OOP) language, and cannot directly generate SQL. The ORM component allows data to be serialised and de-serialised from the database in an automatic fashion. The programmer no longer manipulates a database, instead manipulating the object model of the system.

Database design Like many databases, a standard relational database is being used to model the logical entities which are being managed in EUROCarbDB. The design of the database for EUROCarbDB needs to be flexible so that it can handle the wide variety of data that will be entered into it. In general, four key logical entities were identified to be modelled: Biological contexts, structural information, reference information and experimental data. A biological context is the biological information related to a particular glycoconjugate. This includes the taxonomy, tissue, and disease state that the molecule was found in. Structural information largely covers the topology of the glycan component of the glycoconjugate, including compositional information and cross-links to other databases. Molecules contained within the general public database will be related to structures published in journals, and so reference information needs to be stored in the database. The final entity to be modelled in the database design is experimental evidence that is used to support assertions that structures are found in their respective biological contexts.

Glycoconjugate The glycoconjugate represents the aglyca part of a complete glycoconjugate molecule. The controlled vocabulary for the glycoconjugate is a key chosen in an external database that is appropriate for the type of glycoconjugate that is being represented. For example, a UniProt identifier can be used to manage the protein vocabulary.

Taxonomy The taxonomy of the biological context that the glycoconjugate in question is found in. A record in the taxonomy table is simply a representation of an entry from the National Center for Biotechnology Information (NCBI) taxonomy database.

Disease The disease state associated with the biological context. Each record here is linked to a disease entry from the Medical Subject Headings (MeSH) vocabulary.

Perturbation A chemical perturbation that has been applied to the biological context.

Glycan sequence The glycan sequence for this particular glycoconjugate. The glycan sequence is stored in the Glyco-CT format.

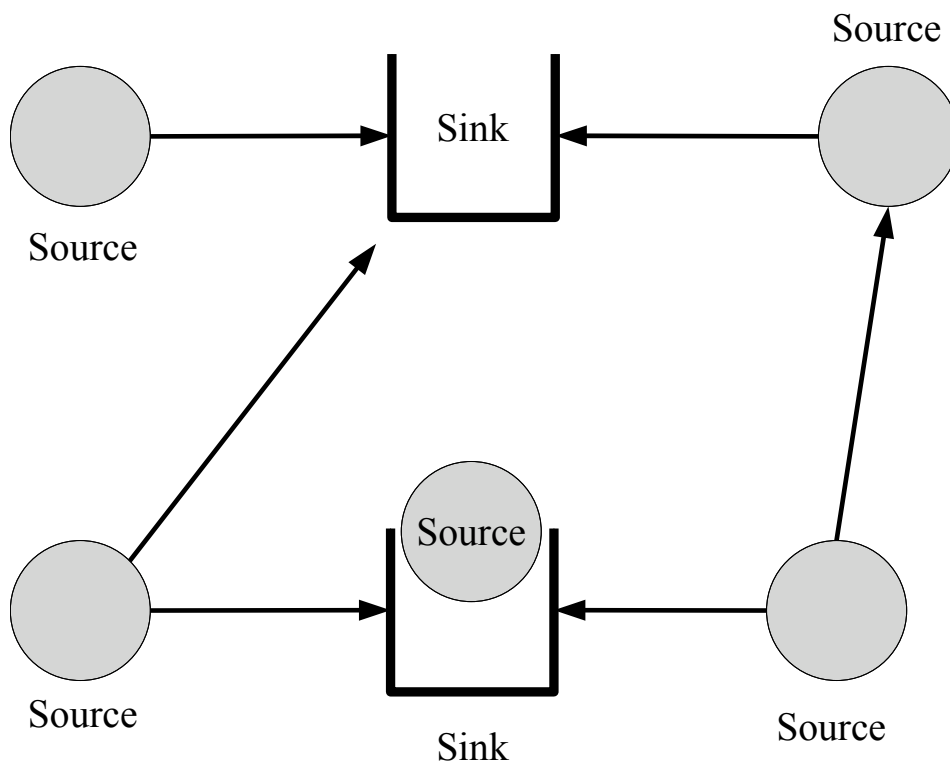


Figure 2: The overall network architecture of the system. Each source and sink is a node within the network, and the arrows represent interactions between nodes. Sources conceptually act as providers of information onto the network, while sinks consume information on the network.

Experiment Experimental data associated with this glycan and biological context. The experimental records usually refer to specific sets of experimental data found in separate sections of the database.

Reference Reference data for this particular glycoconjugate entry. Each record here is indexed by the Medical Literature Analysis and Retrieval System Online (MEDLINE) identifier.

4 Network design

4.1 General network layout

Nodes Each site participating in EUROCarbDB will have at least one node on the network. Each node in the network can have different levels of functionality,

depending on the capabilities that the host institution wishes to expose. As shown in Figure 2, depending on the network layer the node participates in, the node can be labelled as a data source, data sink or both. Data source nodes provide data onto the network via the data layer, whereas sink nodes provide query facilities over the query layer.

Node layout The network is laid out in an undirected mesh layout, with each node able to make connections directly with other nodes as required. The initial network will be a closed network - each node being aware of the other nodes a priori - however the network will eventually expand to accept other nodes onto the network in a dynamic fashion.

4.2 Network layers

The design of the network is split into two basic layers:

- Query layer
- Data layer

Query layer This layer allows queries and control signals to be distributed to the component nodes in the network and then handle the collation of results. The query layer has been designed for most ease traversing firewalls and security procedures in place, and returns results in well defined XML formats.

Data layer This layer allows the database to send raw data between nodes in the form of core database updates, or to distribute experimental data between nodes. The design of this layer is optimised for transferring large blocks of data at irregular intervals.

4.3 Data formats

Usage of XML Import and export formats of data are critical to the application. An XML based format, to ease parsing of data has been decided upon as the basis for all import and export of data. XML provides lots of options for direct transformation and parsing of the data, without requiring a proprietary set of libraries for getting access to the data.

Where possible, the XML output format has been based upon known standards. In general, the XML formats are quite simple, and are only important for associating words from the controlled vocabularies together. If words are used from other controlled vocabularies, they are marked out in the XML files, and provided only as supplementary data.

Each XML file should be validated against a versioned schema. Schemas allow documents to be validated to ensure that the structure and certain aspects of

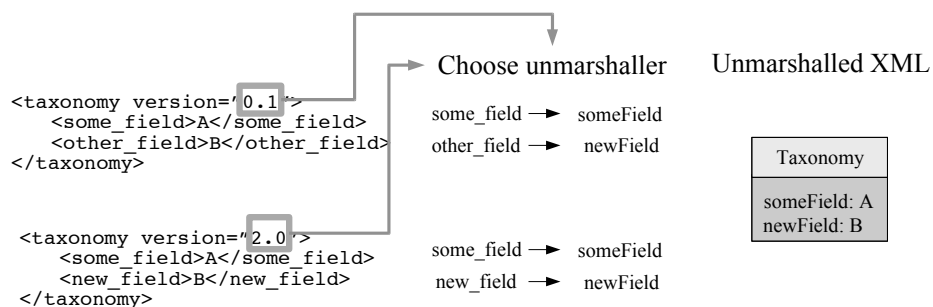


Figure 3: The process by which data is unmarshalled from an XML file. An XML file is received, and the version number extracted from the top-level element in the document. This is used to select an unmarshalling scheme by which the elements in the XML document are mapped to fields within an object. In this example, the `other_field` element has been renamed to `new_field` in the document schema. Since the data being represented remains the same, only the mapping file needs to be modified to reflect this change. Both documents result in the same unmarshalled object.

the content are correct. By versioning schemas, the format of the documents can change, but support for the older data sets can be maintained. Each xml document will have the version number of the schema embedded in it, which will allow for unambiguous identification of document versions. This process is illustrated in Figure 3

Binary formats In addition to using XML and text-based formats, binary formats may need to be exchanged over the network, due to the binary formats used by a number of mass spectrometry vendors.

4.4 Query layer

The query layer is critical to allowing EUROCarbDB to function in a distributed manner. As well as allowing for distributed queries, this layer has several functions involving administration and signalling between nodes on the network. In general, the query layer provides a wrapper around the execution model of the application, exposing the actions as services. The query layer provides facilities to ensure the database can function without a central node, distributing queries to appropriate nodes in cases where the node itself cannot fulfil a query.

4.4.1 Protocols

An asynchronous protocol is required for the operation the query layer. A job is defined as the process of executing an action. Asynchronous dispatch of a job to

actions forming the query layer is required, as the job may be long running. There are number of web-service protocols which can be used for this purpose:

REST REpresentational State Transfer (REST) is a simple protocol for the exchange of messages, modelled closely on the actions of SOAP. The protocol handles a limited set of operations acting upon well defined resource types. This protocol is generally more useful as a web services protocol rather than a general purpose peer to peer messaging protocol.

XML-RPC XML Remote Procedure Call (XML-RPC) uses XML messages as the basis for all messages, and HyperText Transfer Protocol (HTTP) as a transport layer for messages. XML-RPC is a simple protocol, and cannot handle some of the sophistication in the query layer that is required for EUROCarbDB. Although asynchronous calls are not supported within the specifications for XML-RPC, it is possible to add support through API protocol design.

SOAP SOAP is a family of recommendations for the use of an XML-based protocol for message passing and remote procedure calls. Based upon XML-RPC, SOAP was extended to handle more sophisticated use cases, and be more generally applicable as a web-services protocol. SOAP is the base protocol for the query layer for EUROCarbDB due to its ability to support asynchronous calls as part of the protocol, and its support for extensions in the area of security.

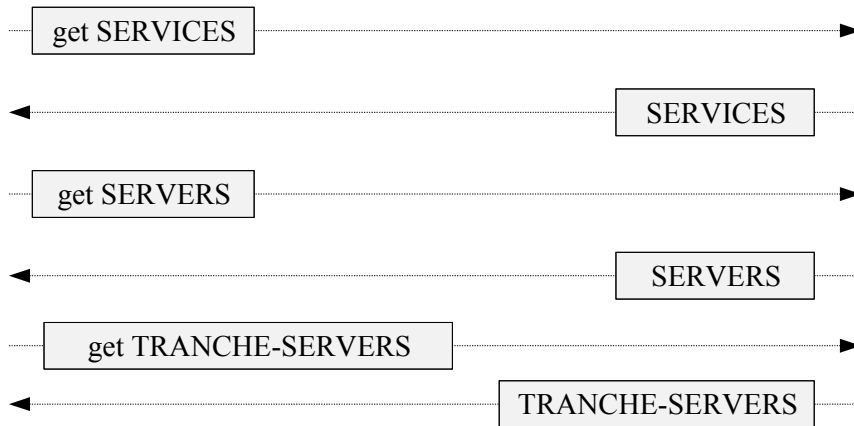
4.5 Messages

There are two types of messages which will be passed between nodes on the network. The first family of messages are control messages, intended to get information from a node about the current state of the network, and the capabilities that it provides to the network. The second family of messages detail the message conversation protocol between nodes when one node requests another node to perform an action.

4.5.1 Control messages

Control messages generally have a simple conversation protocol between sender and receiver. A control message request is generally in the form of asking the target server for a piece of data such as the services it provides, list of known servers etc. These messages do not require authentication as the message content is public information, and is required for an un-authenticated server to join the network. A set of messages from the two families of messages can be found in Figure 4.

Control messages



Query messages

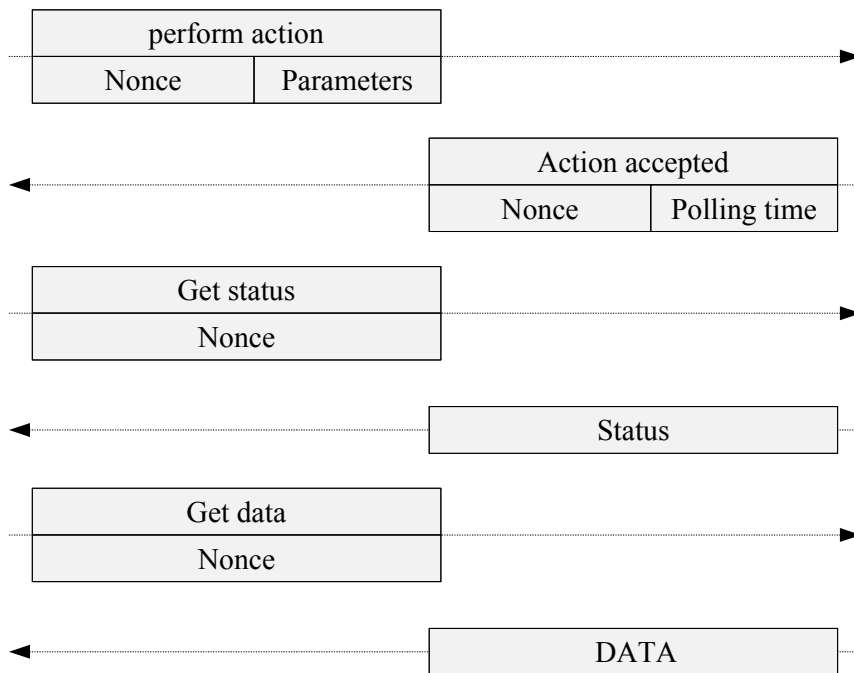


Figure 4: Message conversation between nodes with control and query level messages. Control messages are generally simple messages requiring no authentication and providing publicly accessible information. Query messages require some more security, and involve the use of authentication and keys to prevent attacks on the communication.

4.5.2 Query messages

Query messages have a more complex protocol for node to node conversation. Since the nodes require authentication, various security measures need to be put into place. A Nonce is a key string which is used to ensure that the messages sent between two trusted entities is not replicated and used in replay attacks. When one node dispatches a job to the other node, it attaches the Nonce to the query message. The nonce acts as a secret key for this particular job transaction, acting as a job identifier. The accepting node keeps track of the job identifier and returns a message which indicates the status of the job, a polling time, a new job status identifier and the original job identifier so that the originating node knows which job to keep track of. The originating node waits for the polling time to elapse, and then polls the accepting node. The accepting node returns the status of the node. This polling loop continues until the query is complete and the status indicates this. The querying node then queries the accepting node using the job status identifier, and the data is returned.

4.5.3 Capability discovery

Capability discovery is one type of control message. When a node becomes aware of another node on the network, it should be aware of the services provided by that node. A request for the services available from the node yields a global list of services available on all the nodes. The list of services is an enumeration of all the actions provided by a server, and can be formatted simply as a list of action names, or as a Web Services Description Language (WSDL) document.

4.5.4 Message authentication

Different users should have different access levels on the network, depending on the action they are performing. In a distributed environment, it is not possible to know the usernames of all the users accessing the services. For this reason, a traditional username and password authentication system for web services is insufficient. Instead a system needs to be established so that for each message accepted by a node, the user needs to be uniquely identified and the permissions of the user resolved. Figure 5 illustrates the authentication mechanism.

Each group has a private key. Users are assigned to groups. Each action knows the public keys of the groups it has access to. When a user wishes to perform an action, it sends a request to the node with the group memberships. The node then returns an encrypted nonce, encrypted using the public keys of the minimum set of groups that the user must be a member of to access the action/data. The user then decrypts the nonce, and uses that as a key to access the results of the action.

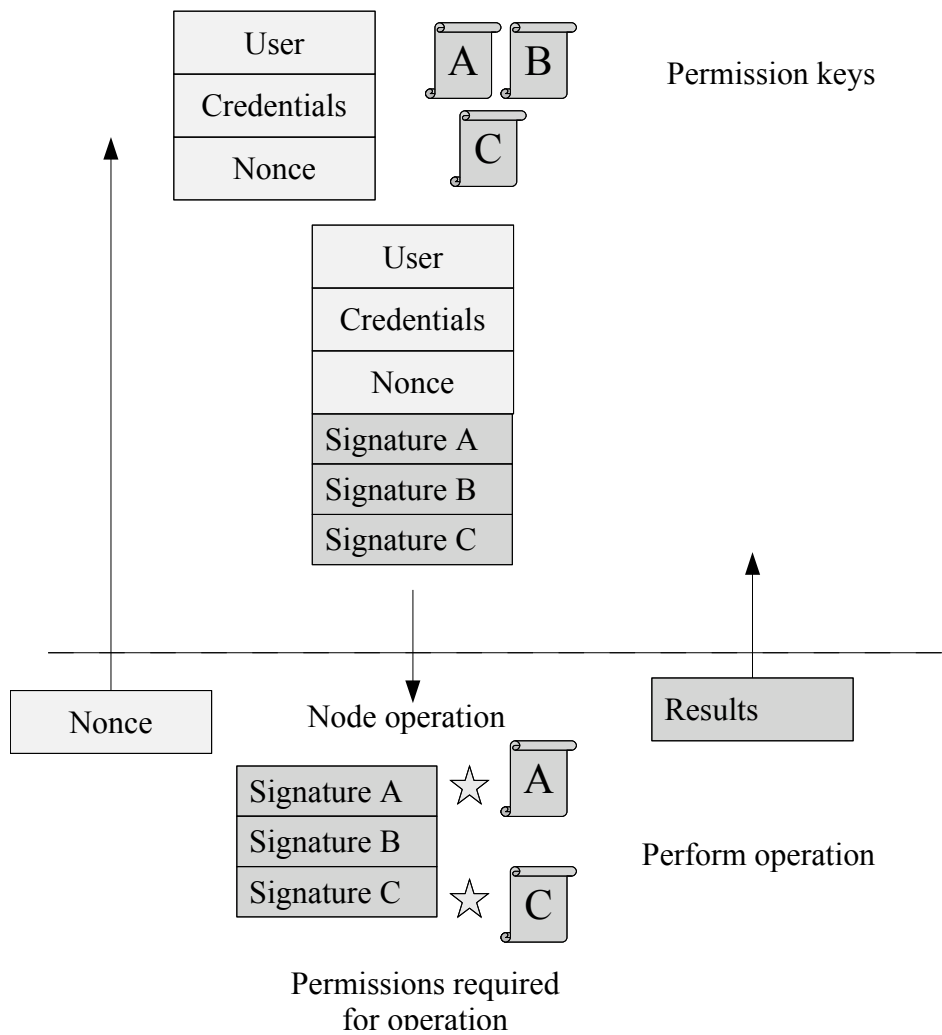


Figure 5: Authentication process for untrusted users from remote nodes. Nonces are signed using public certificates to ensure that the user is in possession of the rights to perform the action. If the remote node - including the associated user - and the local node both have access to the permission certificate, then the remote node can be thought of to have permission to perform the action on the local node.

4.5.5 Action messages

Action messages regulate the execution of actions on remote nodes. When considering the timeline of action execution, we can assume that the user that the action runs as has been authenticated with privileges as described in Figure 5.

Parameters Different actions will require different data types to be passed to them. Since all the different data structures that will eventually be used in the system cannot be known beforehand, a loosely typed parameter set is being used. This is conceptually a keyed set (i.e. a hash) of the different parameters required.

Error states For robust operation of the network, error states need to be well defined. There will be two classes of runtime error for the actions - faults and exceptions. Faults are non-fatal errors which have occurred during the processing of the action, and are appended to the result. These non-fatal errors affect the resultant data, but do not halt the operation of an action. The exception class of errors result in the termination of execution of the action. Errors cannot be recovered from by user intervention, and are usually suggestive of deeper problems within the software. Fault, and to some extent, error codes are largely dependent on the action authors, and their specific types will be determined by the work that the action does. Each action would need to thoroughly document the fault and error codes to provide help to users and administrators of the action.

Result data All actions should be concerned with the manipulation of the data model entities found within EUROCarbDB, with the result of the action being a modified set of entities. As such, the result data stored within a message should be an XML representation of the data that has been manipulated. Result sets should be contained in a simple container element - usually named in accordance with the action that produced the data. For example, the action `show_tissue_taxonomy` can produce results with a top level XML element of `show_tissue_taxonomy_result`. Each entity is tagged with a unique identifier which will allow for the collation of results from different nodes.

4.6 Data layer

The data layer is concerned with the distribution of primary data amongst nodes in the network - including the distribution of core data along the network and the distribution of experimental data amongst nodes.

4.6.1 Protocols

A number of protocols optimised for the transfer of large blocks of data were evaluated on their suitability for use in this layer of the network. A large factor in choosing the protocol to use on this layer is the practicality of the protocol. Real

world security concerns preclude the use of certain protocols, and so the Tranche protocol was chosen due to the high likelihood of administrative acceptance.

BitTorrent Bittorrent is a peer-to-peer protocol for sharing of files created by Bram Cohen. It is a protocol designed to ensure high availability and speedy distribution for a file which is placed on a bittorrent network. The protocol itself is sophisticated, and efficiently routes data to peer nodes on the network. Due to the effectiveness of the protocol, it has gained popularity in file sharing communities, and is often used for the illegal distribution of copyrighted material. Bittorrent network traffic comprises an estimated 55% of the traffic on the internet. Due to the popularity of this protocol for this activity, network administrators have been finding ways to throttle the traffic from this protocol, either reducing the efficiency or blocking the action of the protocol altogether. This could conceivably raise issues in university environments, as special administrative exceptions would be required to enable the installation of the software. Indeed, for maximum efficiency, the node itself should be an equal peer on the network - with no firewalls or restrictions on communication between the peers. This may not be possible in some university environments.

Tranche The Tranche distributed file system is a peer to peer system for the dissemination of large data sets in the proteomics area. The system is based upon a block-based distribution of data amongst participating nodes in the Tranche Distributed File System (DFS). In addition, a Public Key Infrastructure (PKI) security system is used to authenticate users, and encryption is optionally available to users to encrypt their data files on upload. The system is further explained in Section 4.6.2.

Custom protocol An alternative to using a library for the network component is to base the data sharing component upon the messaging found in the query layer. Binary data can be encapsulated within an XML envelope, and then distributed amongst the peers on a request basis. However, the text-based approach to this communication will have a significant overhead, and would not be suited well to the transfer of binary data.

4.6.2 Proteomecommons Tranche network

The ProteomeCommons Tranche network is a peer-to-peer application and network for distribution of data primarily related to proteomics. The network is a secure network, using PKI to authenticate users. The system aims to distribute data by storing data redundantly across the nodes. Although the storage of data itself is decentralised, each node needs to have full knowledge of the network to retrieve data in the most reliable fashion.

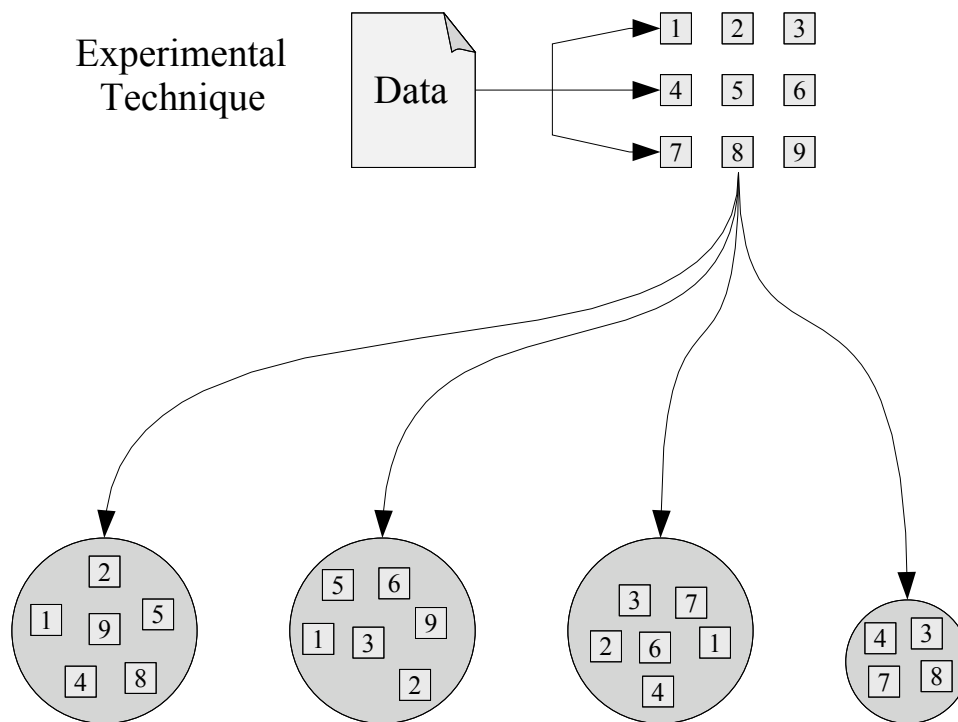


Figure 6: Distributing a file using the Tranche network. Data is given to a peer in the network, which then fragments the data and distributes it to the other peers in the network. Not all nodes on the network are equal, and have different capacities for storing files, resulting in differing amounts of file blocks accepted. By redundantly distributing the blocks of the files to different nodes on the network, availability of the data can be ensured.

Distribution of data When a data file is placed upon the network, the node initially accepting the data breaks it up into smaller blocks (see Figure 6). The accepting node has full knowledge of the other peer nodes, and then distributes these blocks to the other nodes on the network. In this way, each node in the network has part of the whole file. By redundantly distributing the data across as many nodes as possible, it can be ensured that the data remains accessible if any node is removed from the network. If a node is removed from the network, there is a high probability that other nodes on the network will have a copy of the particular blocks which were present on the no longer functional node.

Authentication Like EUROCarbDB in general, the Tranche network requires the ability to authenticate users in a decentralised manner. Signed certificates are used as the basis for establishing the identity of any user on the Tranche network. The certificate mechanisms are further explained in Section 4.8.

Academic support EUROCarbDB can take advantage of the libraries and existing network of Tranche. Since ProteomeCommons.org is already working to gain acceptance in the academic arena, this work will not have to be done by EUROCarbDB. Proof of the security is tantamount to the acceptance of any system, and by taking advantage of the work done by the Tranche developers, much of the administrative and political issues surrounding the installation of the servers will be overcome.

4.7 Data merging

Maintaining the integrity of the database across all the peers is a crucial factor for maintaining the quality of the database. Rather than requiring full availability of nodes to all other nodes on the network to maintain uniqueness, a synchronisation process has been designed to maintain the integrity of the database across all nodes.

4.7.1 The need for merging algorithms

Within any database, the structure of the data can be recognised as a graph, where each of the nodes in the graph are rows in tables, and the edges are relationships between rows. For a database that is distributed, one of the largest challenges is maintaining unique rows across all the different entities, partitioning the graph so that nodes are shared between different segments.

For this reason, local and global identifiers are used to identify entities in the database. Local identifiers are used to maintain relationships between entities, while global identifiers are used when entities are referenced outside of their originating server.

4.7.2 Synchronisation process

Synchronisation of the database cannot happen in a completely decentralised manner. There needs to be a node which acts as the controller of the process. In order to choose the controller, an election process can be used to determine the node to be marked as the master node. For the purposes of this prototype, the election process is not required, and one node will be nominated as the master node.

Synchronisation of the database is done sequentially, with the elected master node co-ordinating the process. Each node in the database is sequentially synchronised with the elected master. This process declares that the elected master has the reference copy of the core database.

4.7.3 Data canonicalisation and hashes

In order to easily compare entities, a unique hash code for each entity has to be obtained. This hash code can be calculated from combinations of fields which are guaranteed to be unique, and come from a controlled vocabulary of some sort. Simple hashing codes could be things like UniProt IDs, NCBI taxonomy ids, or the Glyco-CT sequence. It is important that the hash-codes remain unique, as they will be used as a global identifier for a particular entity.

4.7.4 Data updates from core

Figure 7 illustrates the process of synchronising a local database with the updated core database. Any new public entities on the master core database need to be added to the local databases so that they may be referenced by any new records. Where there is no local copy of the entity, the process is simple. The new entity is simply added to the local database, given a local identifier, and will keep the global identifier to indicate that it has already been synchronised with the core database.

4.7.5 Parallel insertion of duplicate entities

A more complicated case is shown in Figure 8. In this particular case, the local copy of the core database has also added an identical copy of the new core entity into the database. Since the network as a whole is only concerned with the GID of the entity, the GID from the updated core entity from the master is used in the local entity. It is safe to change the GID of a local entity since the referential integrity of the database is dependent only on the LIDs of the local database.

4.7.6 Update from local database to core

New records are not resynchronised to the core database automatically. To transfer a record from the local database to the master core database involves a publishing process for the record.

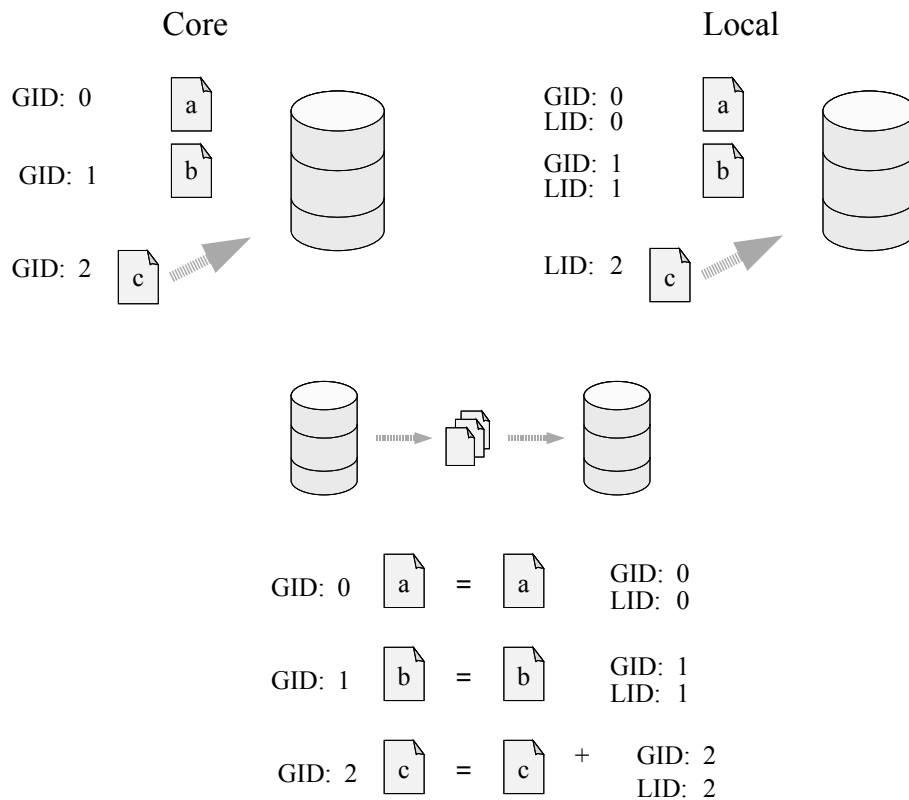


Figure 8: In the event of the master and local databases receiving updates in parallel, the synchronisation process is modified from that shown in Figure 7. The synchronisation process deviates from the former process where the entity 'c' from the master core is compared with the entity from the local core. Since both entities will have the same identifying hash, the local entity accepts the GID from the master core entity.

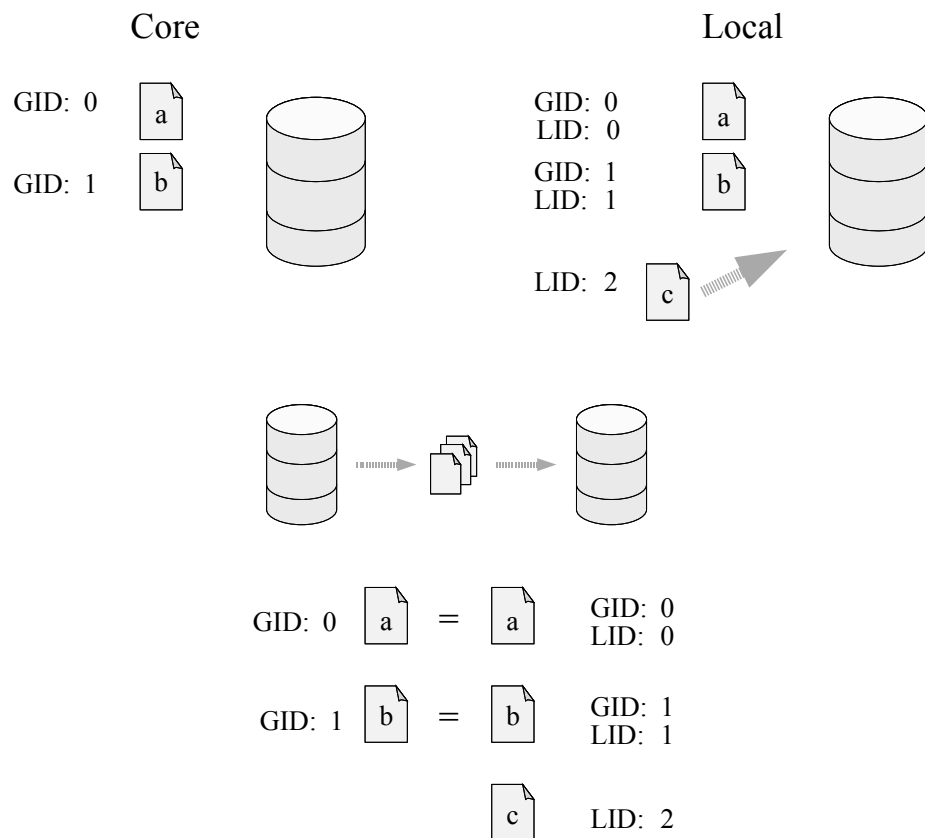


Figure 9: Synchronisation from the local core database to the master core database does not occur using the regular synchronisation methods. A special operation involving the explicit publishing of records is required. Since entity 'c' does not have a entity hash equal to any of the other entities from the master core, it does not require the synchronisation.

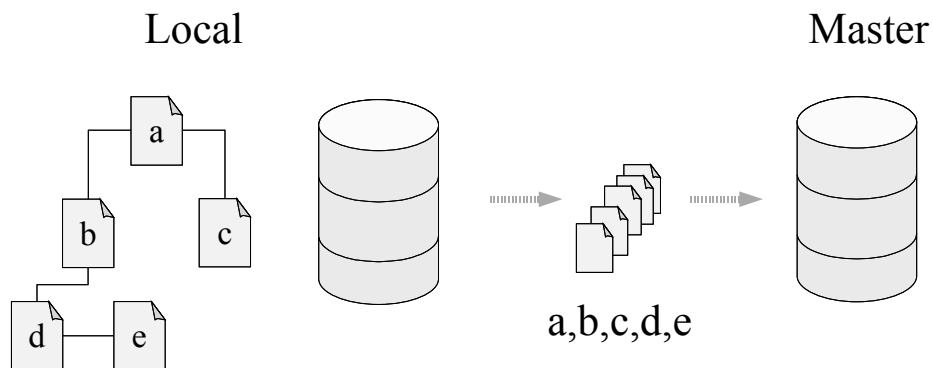


Figure 10: Publishing data to the master node. A user decides that they wish to publish a set of entries. The minimum entity set which is required to accurately represent the entries which need to be published is calculated. Each entity is then serialised and sent to the master node to be added to the core as a local insertion of data. This new version of the core database can then be resynchronised with the other nodes in the network.

A multi-user database present a number of challenges with respect to maintaining the database after updates to data. Data can be updated when the record has been erroneously entered, or the annotations on the data has been corrected. If other users have taken this data as the basis for a record they have entered, it would be incorrect to update the previously entered data, as the records of other users may be also affected. For this reason, only creation actions on commonly used core components are allowed. Updates are achieved by duplicating the incorrect record while fixing the incorrect information.

Deletion of records will only be allowed by the nodes which have ownership of the record. On receiving updates, notification of any deleted records is sent to the master node, and then distributed to the other nodes on the network. Each node on the network can then decide whether they wish to delete the record locally.

4.7.7 Publishing of records

In order to move records from the local database to the core database, a record set is identified by the user as being ready for publication in the general database. The data is exported, and then imported into the elected master node. After a record set is imported to the master core database, the core database is then synchronised back to the full set of peer nodes.

4.8 Security

Security of both the application and the data that is transmitted across the network is very important to the operation of the network. We can consider that the network

will be run across a public and insecure network, and so another layer of encryption should be used to ensure that the data remains safe.

4.8.1 Public key cryptography

Public key cryptography is a mechanism by which documents can be encrypted so that only people who have possession of a secret key can decrypt the data. The premise of public key cryptography is to use a pair of cryptographic keys, one public and one private. A key is essentially a long set of bits which is used to transform a document from plaintext into an encrypted form. The choice of public and private keys is important, as the private key should be able to be used to decrypt the cyphertext resulting from a public key encryption. The strength of public key encryption comes from the mathematical difficulty of finding factors of a number. While it is possible to easily verify whether the product of two numbers are equal, it is significantly more difficult to find the two original numbers via factorisation. A number of algorithms can be used to generate the public and private key pairs, but one of the strongest is to use the RSA algorithm [9].

4.8.2 Digital signatures

To digitally sign a document, a hash of the document is first calculated - essentially the fingerprint for a particular document. This fingerprint is then encrypted by the author of the document using their private key. This encrypted fingerprint is the signature for the document. The document and signature are then sent to the receiver. The receiver can then try decrypting the fingerprint using the public key of the document sender. By comparing the fingerprints of the actual received document, and the decrypted fingerprint, it is possible to ascertain whether the document was modified in transit.

4.8.3 Certificate mechanisms

One of the key challenges in a distributed setup is to ensure that the endpoints on the network are actually who they claim they are. One of the standard ways to establish identities of nodes is to use a certificate mechanism. The X.509 certificate mechanism allows for a web of trust to be established so that nodes can be identified. A certificate is an identity document - usually identity information and the public key of the entity being identified - which has been signed by a certifying authority. If the user implicitly trusts the certificates of the certifying authority, then any certificate signed by the certifying authority can be thought to be verified too. The combinations of encryption schemes, signatures and certification mechanisms can ensure that the data being sent across the network is verifiable.

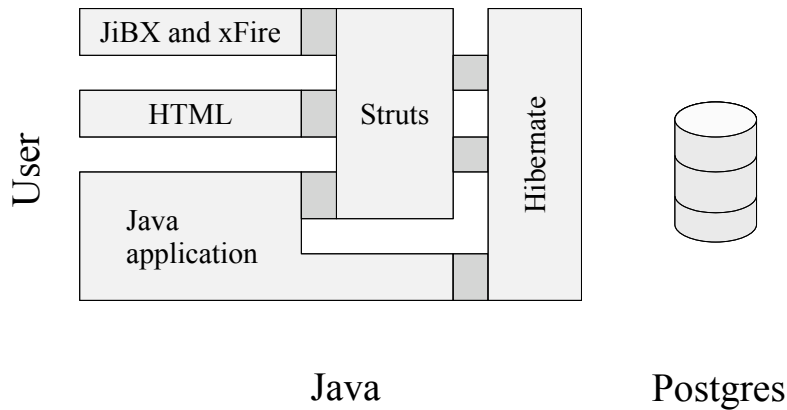


Figure 11: Relationships between the software components in EUROCarbDB. This diagram illustrates the software components for the various architectural components shown in Figure 1. The architecture is a Java and SQL database based system, where the Struts and Hibernate components form significant reused components in the architecture.

5 Software implementation

The application is implemented as a first prototype using a three tier application environment - operating on the Java [10] platform, using the Hibernate [11], Struts 2 [12], Tomcat [13] and PostgreSQL [14] libraries to add the desired functionality to the basic server. In addition, JiBX [15] as well as XFire [16] are used to supply the XML functionality. In general, open source libraries and applications are used as the basis for the entire application. An overview of the general software application architecture can be seen in Figure 11.

5.0.4 Java and Tomcat

The Java language is a bytecode compiled language, known for the ease of cross-platform development and wide availability of libraries. The Apache Tomcat application server provides an environment under which Java web-based applications can be offered to the public. Both the Tomcat server and Java are open source, albeit licensed under different schemes (Apache 2.0 License and GNU Public License (GPL) v2 respectively).

5.0.5 Hibernate

Hibernate provides an ORM layer onto the database by reverse engineering a set of Java sourcecode files from the database schema. By reverse engineering the Java

API from the database schema, there is no need to synchronise the changes between system models, the database model and object model. As the reverse engineering process is a somewhat naive process, the reverse engineered source code needs to be modified so that an appropriate API is exposed for manipulating the objects.

5.0.6 PostgreSQL

PostgreSQL is used as the database component in the application. The optimal way to store data for the application is in a relational database, and the PostgreSQL database provides the best set of features and performance of the open source databases. Connection to the ORM component, and the Java applications in general is achieved through the use of JDBC libraries.

5.0.7 Struts 2

Web requests are essentially stateless operations. Each request does not know about the requests which have occurred before it, and there is nothing within HTTP to explicitly bind a series of requests together in a transaction. For this reason, support for saving state has to be added to the server or client side. Server side state is established by using cookies - identifiers which are stored on client machines - which can tie a particular request to a particular server state. Many changes to state can occur for a request - such as a user logging in, or being part-way through a multi-stage action. Since a lot of this functionality is common to many web applications, a number of frameworks have been developed to provide these common functions. One such framework is the Struts framework. Struts follows a MVC approach to the development of web applications, providing a set of Java classes which provide the basic functionality within the application.

Core to the function of the Struts is the concept of an Action. Source code associated with an action is related to a single unit of work, or a verb, which is applied to the system. Simple actions can be "Add biological context", "Associate Taxonomy", "Upload Experimental data". Each action should be atomic, and should not be dependent on the state of other actions. Struts manages the execution of the actions, as well as executing a series of pre and post-execution methods as interceptors.

5.0.8 XML libraries

JiBX and XFire together provide support for SOAP within EUROCarbDB. In order to save the duplication of work, JiBX was used as the binding library for both SOAP and for generic XML marshalling and unmarshalling. JiBX functions as a bytecode manipulation of classes, marking out the binding of fields in an object to elements in an XML file. The XFire library is an implementation of the SOAP protocol, and handles the dispatch of SOAP messages to appropriate methods. A specialised SOAP message to Struts action wrapper was written to allow the use of all actions as SOAP methods.

References

- [1] Amos Bairoch, Rolf Apweiler, Cathy H Wu, Winona C Barker, Brigitte Boeckmann, Serenella Ferro, Elisabeth Gasteiger, Hongzhan Huang, Rodrigo Lopez, Michele Magrane, Maria J Martin, Darren A Natale, Claire O'Donovan, Nicole Redaschi, and Lai-Su L Yeh. The universal protein resource (uniprot). *Nucleic Acids Res*, 33(Database issue):D154–9, 2005.
- [2] Kosuke Hashimoto, Susumu Goto, Shin Kawano, Kiyoko F Aoki-Kinoshita, Nobuhisa Ueda, Masami Hamajima, Toshisuke Kawasaki, and Minoru Kanehisa. Kegg as a glycome informatics resource. *Glycobiology*, 16(5):63R–70R, 2006.
- [3] Thomas Lutteke, Andreas Bohne-Lang, Alexander Loss, Thomas Goetz, Martin Frank, and Claus-W von der Lieth. Glycosciences.de: an internet portal to support glycomics and glycobiology research. *Glycobiology*, 16(5):71R–81R, 2006.
- [4] F.V Toukach and Yuri Knirel. New database of bacterial carbohydrate structures. In *Proceedings of the XVIII International Symposium on Glycoconjugates, Florence, Italy*, pages 216–217, 2005.
- [5] Glycominds website (<http://www.glycominds.com>).
- [6] S Doubet, K Bock, D Smith, A Darvill, and P Albersheim. The complex carbohydrate structure database. *Trends Biochem Sci*, 14(12):475–477, 1989.
- [7] C A Cooper, M J Harrison, M R Wilkins, and N H Packer. Glycosuitedb: a new curated relational database of glycoprotein glycan structures and their biological sources. *Nucleic Acids Res*, 29(1):332–335, 2001.
- [8] Tranche website (<http://tranche.proteomecommons.org/>).
- [9] R. L. Rivest, A. Shamir, and L. M. Adelman. A METHOD FOR OBTAINING DIGITAL SIGNATURES AND PUBLIC-KEY CRYPTOSYSTEMS. Technical Report MIT/LCS/TM-82, 1977.
- [10] J Gosling, B Joy, and GL Steele. *The Java Language Specification*. Addison-Wesley Longman Publishing Co., Inc, Boston, MA, USA, 1996.
- [11] Hibernate website (<http://www.hibernate.org>).
- [12] Struts website (<http://struts.apache.org/2.x/>).
- [13] Tomcat website (<http://tomcat.apache.org/>).
- [14] Postgresql website (<http://www.postgresql.org/>).
- [15] Jibx sourceforge (<http://jibx.sourceforge.net>).

[16] Xfire sourceforge (<http://xfire.sourceforge.net>).