# The gRNA: A Highly Programmable Infrastructure for Prototyping, Developing and Deploying Genomics-Centric Applications

AMEY V LAUD[1] SOURAV S BHOWMICK[2] PEDRO CRUZ[1] DADABHAI T SINGH[1] GEORGE RAJESH [1]

HeliXense Pte Ltd [1],
73, Science Park Drive,
# 02-05, Science Park 1,
Singapore 118254
{alaud, pcruz, dtsingh, george}@helixense.com

College of Engineering[2],
School of Computer Engineering,
Nanyang Technological University,
Singapore 639798
assourav@ntu.edu.sg

## Abstract

The evolving challenges in lifesciences research cannot be all addressed by off-the-shelf bioinformatics applications. Life scientists need to analyze their data using novel or context-sensitive approaches that might be published in recent journals and publications, or based on their own hypotheses and assumptions. The genomics Research Network Architecture (gRNA) is a highly programmable, modular environment specially designed to invigorate the development of genomics-centric tools for life sciences-research. The gRNA provides the development environment in which new applications can be quickly written, and the deployment environment in which they can systematically avail of computing resources and integrate information from distributed biological data sources.

## 1 Introduction

There is an increasing impetus in using computer-based ('in-silico') techniques for modeling biological behavior based on genomic and related information. As with most scientific research, experimental approaches in genomics are not initially validated by or have overall consensus within the entire scientific community. To scientists working on such approaches, development of applications ought to be possible without needing considerable expertise in software-development. Likewise, to software engineers participating in lifesciences research, there ought to be clear abstractions from the biological world that can be used as building blocks.

When developing new functionality in bioinformatics, scientists need to consider the following key issues:

- Most bioinformatics research rely on a combination of a wide set of related public and private databases [19]. These databases can contain *raw* information referring to the sequenced genomic code of organisms [11], or the results of new high-throughput techniques such as microarrays [25], or *curated* databases containing carefully scrutinized existing research, systematically compiled by domain experts [33]. It is useful [17] to correlate these databases with those containing medical records [32], information on disease [28], databases on references to literature [10], databases containing information on the properties of chemicals and their molecular structure [40]. Therefore, useful queries do span through a multitude of databases in the categories mentioned above [27].
- Most data pertaining to genomics and molecular biology is characterized by highly evolving, semi-structured data [23]. Moreover, genomic data available from public and private repositories is voluminous [34], highly heterogeneous, and not consistently represented. The heterogeneity results from our evolving understanding of complex biological systems; the numerous disparities in modeling biological systems across organisms, across tissue, in different environments and over time; and disparities across the scientific community in their understanding of these systems.

**Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002**

Moreover, there continues to be a lack of common standards in the representation of biological data [6].

- Many of the publicly available or commercial databases are the subject of detailed, often subjective licensing conditions that the developer must take into account [16].

- Successful data management and integration is only part of typical bioinformatics problem-solving. Productive and extensive application development in the field is also hindered by the scarcity of 'bioinformaticists' [41] - a term used to refer to professionals that are equally proficient in computer programming and in one or more biological domains including genetics, molecular biology, genomics and biochemistry. A majority of application development must be guided by the combining the expertise of domain experts and computer scientists as a team.

- Bioinformatics applications rely heavily on a combination of basic techniques ranging from the analysis and comparison of genomic sequences [35], gene-expression analysis [18], analysis of molecular structures and common statistical techniques such as clustering and hidden Markov models. In terms of providing domain-specific functionality, there exist such programmable frameworks as BioJava [1], BioCORBA [2] and the Systems Biology Workbench [14].

- Finally, considering the voluminous nature of much of genomic data, useful implementations of bioinformatics applications should include visual interfaces that can help to provide multi-modal views of the wide variety of biological data and the results of analysis [18].

The *Genomics Research Network Architecture* (gRNA) was designed and developed at HeliXense Pte Ltd, Singapore [5] to address these challenges by assuming each of them as the foundational requirements in new bioinformatics applications. It was developed with the ideology that new genomics-centric applications and algorithms be prototyped, developed, tested and deployed using the same hypothesis-based way of doing research that is characteristic of the lifesciences.

The gRNA provides a *development environment* and a *deployment framework* in which to maintain distributed warehouses, and to model, query and integrate disparate sources of data. It also provides the mechanisms to account for the disparities in the nomenclature and representation of data, as well as the semi-structured nature of most types of data. In the same context, the architecture strives to prevent the introduction or imposition of new biases. By providing useful domain-specific abstractions and functionalities, and related visual interfaces, the gRNA effectively minimizes the time, cost and degree of human expertise typically required in the bioinformatics application-development process. In this paper, we focus our discussion on the following issues:

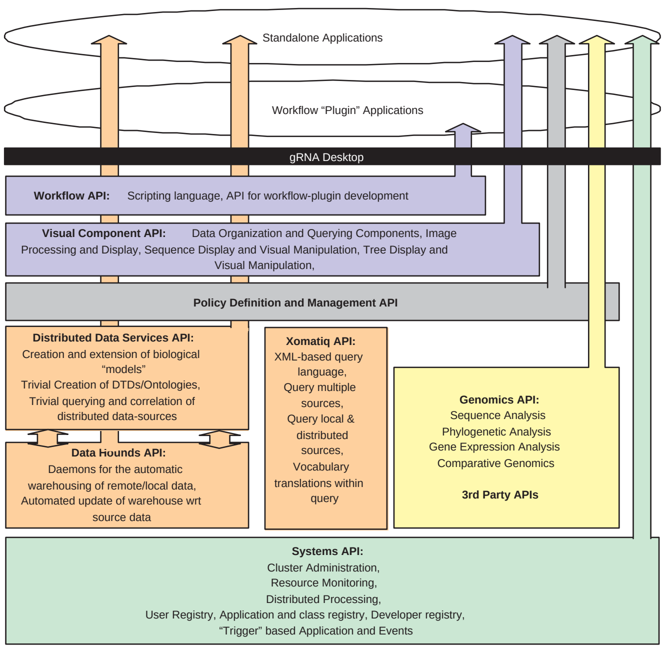

Standalone Applications
Workflow "Plugin" Applications
gRNA Desktop
Workflow API: Scripting language, API for workflow-plugin development
Visual Component API: Data Organization and Querying Components, Image Processing and Display, Sequence Display and Visual Manipulation, Tree Display and Visual Manipulation,
Policy Definition and Management API
Distributed Data Services API: Creation and extension of biological "models" Trivial Creation of DTDs/Ontologies, Trivial querying and correlation of distributed data-sources
Data Hounds API: Daemons for the automatic warehousing of remote/local data, Automated update of warehouse wrt source data
Xomatiq API: XML-based query language, Query multiple sources, Query local & distributed sources, Vocabulary translations within query
Genomics API: Sequence Analysis Phylogenetic Analysis Gene Expression Analysis Comparative Genomics
3rd Party APIs
Systems API: Cluster Administration, Resource Monitoring, Distributed Processing, User Registry, Application and class registry, Developer registry, "Trigger" based Application and Events

Figure 1: gRNA Development Platform.

- We discuss the overall architecture of gRNA for developing and deploying genomics-centric applications.

- We demonstrate the viability of the system using an application constructed using the gRNA facilities.

It is worth mentioning that the key inspirations behind the gRNA methodology have been analogies in other fields that have successfully established themselves as the standard and effective approach to develop new functionality. These include Matlab [8], a programmable framework to quickly develop new mathematical models, Maya [9], a programmable framework used for developing new functionality in 3D animation and visual effects. In both these frameworks, important contributions to the programmable functionality are made by independent parties, which are then made available as programmable interfaces referred to as *plugins* and *toolboxes*.

The rest of the paper is organized as follows: In Section 2, we elaborate on the architecture of the applications development platform of gRNA. We illustrate how to develop applications using the facilities provided by this platform in Section 3. In Section 4, we present the deployment environment of the gRNA. We discuss our work with respect to the existing work in bioinformatics in Section 5. Finally, the last section concludes the paper.

## 2 The gRNA Development Platform

The genomics Research Network Architecture comprises of a development platform in which to prototype and build new applications, and a deployment environment in which to provide access to distributed computing and data resources. In this section, we discuss the development platform in detail. In the next section, we present the gRNA deployment environment. We begin by giving a brief overview of the components of the development platform and then discuss some of these components in detail.

### 2.1 Overview

The development platform consists of application programming interfaces (APIs) designed to provide abstractions for and the foundational basis for new functionalities in bioinformatics applications.

Figure 1 shows an architectural overview of the development environment. The APIs are inter-related, yet decoupled. This means that the APIs can be used, extended and improved independently.

Three of the APIs are focused on organizing, modelling and querying biological data - these are the *Data Hounds API* for transporting, wrapping, storing and indexing external databases; the *Data Services API* to provide object-relational abstractions built from multiple primary sources of data; the *XomatiQ API* to provide a query language to systematically query and correlate multiple warehouses of lifesciences data.

The gRNA consists of two domain dependent APIs — these are the *Genomics API*, which provides specific abstractions and functionalities from the biological world; and the *Visual Component API*, which provides multi-modal displays for displaying biological data and visual building blocks for typical experiments.

The *Workflow API*, independent from all the others, is meant to be a generic, configurable engine to pipeline a set of tasks and data into a unified process.

Within the gRNA, we refer to the following as *resources* - computers, computing clusters, data warehouses, implemented Data Hounds, implemented Data Services, toolkits and Document Type Definitions (DTDs). Accordingly, the gRNA maintains a centralized directory of various resources, their state and properties. A *Policy API* allows for the imposition of ownership and conditions for the use of various resources. This means that owners of resources are allowed to assign such conditions as 'view only', 'read only', 'write' and 'execute' to various resources, where applicable. Owners are also allowed to embed terms and conditions to define the basis of agreement between a third party user wanting to use resource under the given party's ownership.

Lastly, the *System and Grid API* provides access to underlying systemic tasks such as distributed programming and access to centralized directories of available resources.

### 2.2 The gRNA Data Middleware

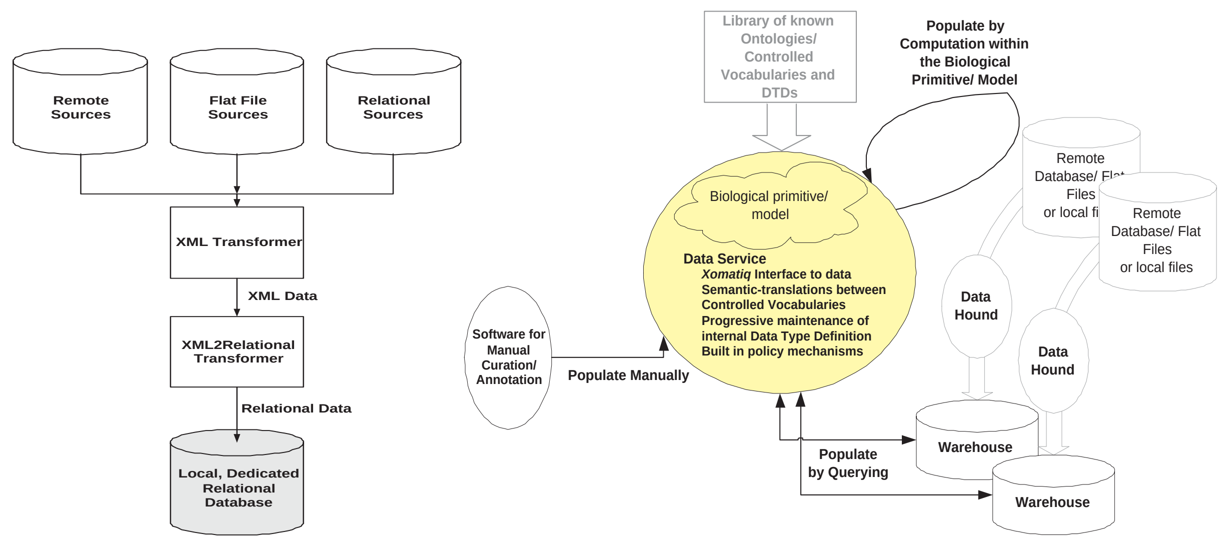
The gRNA Data Middleware is comprised of modular mechanisms called the *Data Hounds*, *Data Services* and a special query mechanism called *XomatiQ*. Because the kernel within the middleware has been designed primarily for semi-structured data management and querying, it is portable to a wide range of application areas that require frequent change in the schema of underlying data. We now describe these three components.

#### 2.2.1 Data Hounds

Apart from the obvious advantages of performance, flexibility and availability, warehousing biological data has a positive impact on the *privacy* of the relevant information retrieval. Users interested in information from biological sources are typically at the mercy of web-sites and service providers that offer public query interfaces [43] to the data. Moreover, many service providers have the rights to monitor and inspect queries formulated on their data repositories [39]. In bioinformatics domain, there is a pressing need to prevent the service providers to predict the objective behind the formulation of a set of biological queries by a user. The *Data Hound API* enables us to efficiently warehouse data locally. It is a third-party programmable mechanisms to facilitate the transport, wrapping and conversion of remotely located relational tables and flat-files into local warehouses that are conducive to semi-structured data management.

Most of the publicly accessible databases of interest are accessible through internet protocols such as FTP (File Transfer Protocol) and HTTP (Hypertext Transfer Protocol). Typically, updates to these databases are also provided through pre-designated locations through the same protocols. Data Hounds must select one of these designated modes for physically transporting data from its original source. Once the transport of the data is complete, the gRNA Data Middleware takes over the storage, indexing and management of the resulting warehouse.

Figure 2(a) shows the functional overview of the Data Hound. First, the *XML-transformer* module harness data (flat files, relational tables, XML data etc.) from disparate biological sources and then convert them to XML data. This involves specifying a set of DTDs for every kind of data in the remote biological sources, and a mapping of the attributes in this data to elements and attributes in the DTDs. For example, one of the commonly used sources of data is the European Molecular Biology Lab's sequence repository


Remote Sources
Flat File Sources
Relational Sources
XML Transformer
XML Data
XML2Relational Transformer
Relational Data
Local, Dedicated Relational Database
Library of known Ontologies/ Controlled Vocabularies and DTDs
Populate by Computation within the Biological Primitive/ Model
Biological primitive/ model
Data Service
Xomatiq Interface to data
Semantic-translations between Controlled Vocabularies
Progressive maintenance of internal Data Type Definition
Built in policy mechanisms
Software for Manual Curation/ Annotation
Populate Manually
Remote Database/ Flat Files or local files
Remote Database/ Flat Files or local files
Data Hound
Data Hound
Warehouse
Warehouse
Populate by Querying

(a) gRNA Data Hound. (b) Data Services - Object-Relational Biological Data Modeling.

Figure 2: gRNA Development Platform.

[4, 43]. This is a large repository that is made available as flat files off its FTP site. Writing the XML-transformer module for this database involves specifying a set of DTDs for the data in the flat-files and a mapping of attributes from these flat-files to elements and attributes in the DTD.

The second component in Figure 2(a), i.e., the *XML2Relational-transformer* converts the XML data to relational tables and store them in the Sybase RDBMS. For example, the XML data generated from the EMBL database is transformed to relations and stored in the RDBMS. Note that this module is motivated by recent research in the area of storing, indexing and querying XML data in a RDBMS environment [31, 37, 44].

Data Hound also has the capability to update the data in the warehouse based on the changes to the remote sources. Once the changes have been committed to the local warehouse, the Data Hound sends out triggers to related application indicating changes to the warehouse.

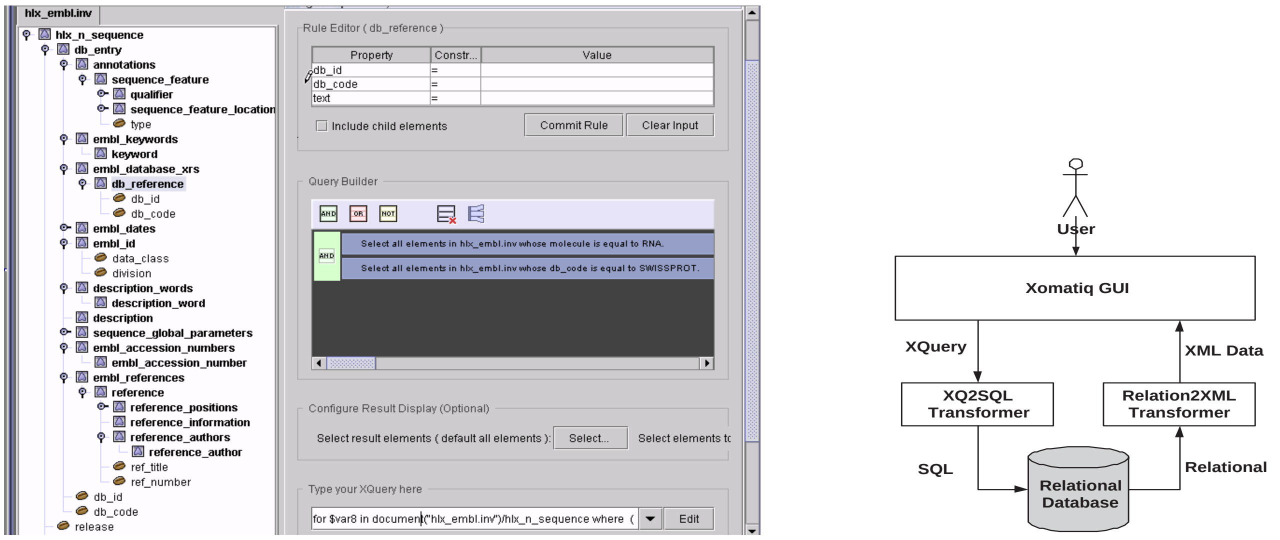
### 2.2.2 Data Services

The gRNA Data Services allow us to treat warehouses as primary sources of data and build more useful, secondary databases based on conceptual object models. Figure 2(b) gives a functional overview of the Data Services.

Each Data Service must derive its initial data structure from existing classes of the Genomics API. As described in Section 2.3, the data structure of all classes in the Genomics API is maintained externally in XML DTDs. This makes it simple for the Data Services to import the DTD of the selected class. Thereafter, Data Services allow incremental additions to be made to this DTD trivially, on an ongoing basis, without disturbing the integrity of data already stored within the Data Service. Moreover, because the DTD is mapped to a known class-specification, Data Services make it feasible to create and then use object instances based on on that class in an application.

As shown in Figure 2(b), a single Data Service is typically populated by making queries to multiple warehouses and then mapping the results to elements in the Data Service's DTD.

Alternately, data can be manually entered into the Data Services. Because of the inherent support for evolving DTD within the Data Service, there is the provision of adding new types of annotations trivially. Objects created from the Data Services are also bound to generate their own data through computational techniques. For example, the developer can choose to create a Human Genome Data Service, to create a conceptually simple object-relational repository of human genome sequences. The *DNASequence class* from the Genomics API provides an abstraction of the DNA Sequence, and has numerous built-in functionalities that allow the developer to initialize it with some DNA sequence data, and then perform a range of sequence analysis and other computations. The developer therefore creates the Data Service based on this class. Once that is done, the user can populate the Data Service by performing queries on the EMBL warehouse, introduced in Section 2.2.1. The user might want to restrict entries within the new Data Service to say, sequences shorter than a certain length. The query on the primary EMBL warehouse would be conducted accordingly. Thereafter, the user would typically look for patterns that indicate the existence of gene regula-



| Property | Constr... | Value |
|---|---|---|
| db_id | = | |
| db_code | = | |
| text | = | |

Select all elements in hlx_embl.inv whose molecule is equal to RNA.

Select all elements in hlx_embl.inv whose db_code is equal to SWISSPROT.

Select result elements ( default all elements ): Select... Select elements tc

```
for $var8 in document("hlx_embl.inv")/hlx_n_sequence where (
```

(a) GUI for XomatiQ. (b) XomatiQ API.

Figure 3: XomatiQ.

tory patterns in some or all of these sequences, something that is usually provided by functionalities within member methods of the DNA Sequence. The results of these computations can be assigned as ‘annotations’ to the respective sequences, therefore meant to be saved persistently. Whereas some of the annotation types might already exist within the schema of the Data Service, others might have to be added incrementally by editing the DTD. The Data Service provides a number of convenience routines to enable editing of its DTD.

Data Services also maintain a listing of known ‘controlled vocabularies’ used to represent biological data and the semantic relationship between these vocabularies, known as ontologies. A mapping between equivalent terms across vocabularies is also maintained. This makes it simpler to conduct queries that dont necessarly specify elements as listed within the database, but use ‘equivalent’ terms that the end-user might be more familiar with.

Another advantage of packaging information into Data Services is that they can be accessed from remote clusters, as long as the user has the necessary rights to accessing the Data Service.

### 2.2.3 XomatiQ

The XML structure of the underlying data makes it conducive to using a XML query language. The XomatiQ API provides an XML-based query language to facilitate the querying of one of more, distributed or local warehouses managed within the gRNA. Querying of distributed warehouses is facilitated transparently using the underlying *Data Grid* facility that is introduced in Section 4.2. The syntax for the XomatiQ language is based on the W3C XQuery specification

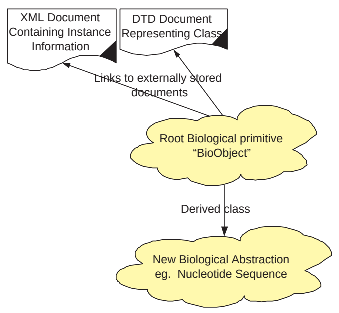
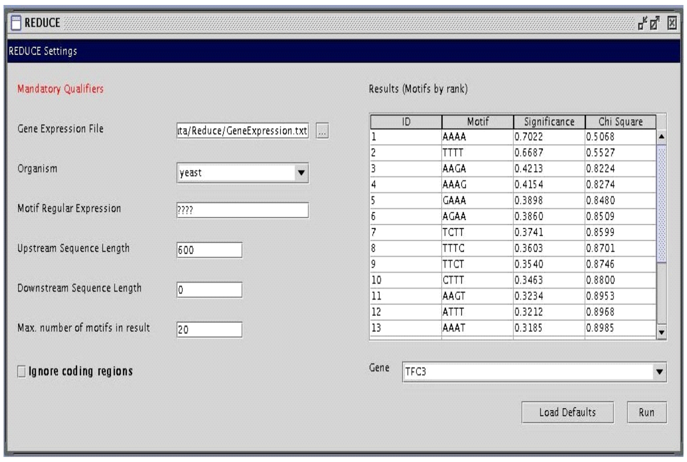

XML Document Containing Instance Information
DTD Document Representing Class
Links to externally stored documents
Root Biological primitive "BioObject"
Derived class
New Biological Abstraction eg. Nucleotide Sequence

Figure 4: Genomics API

[15]. The XomatiQ API extends the syntax of the XQuery specification by making provision for simple unstructured queries, similar to those found in web-based search engines. The extension simply allows the user to specify keywords that are implicitly meant to be located close to one another in the same XML document. This language provides all the functionalities and components required by biologists to analyze their data in a biological meaningful way [36]. A screenshot of the GUI for formulating queries using XomatiQ is given in Figure 3(a).

Figure 3(b) gives a functional overview of the query mechanism. Once a user formulates a query using XomatiQ (XQuery-like language), it is fed to the *XQ2SQL-transformer* module which rewrites the query to corresponding one or more SQL queries. These queries are evaluated against the warehouse where the data is stored in relational tables. The query results returned by the warehouse is in the form of relations. Hence, the module *Relation2XML-transformer* converts the relations to corresponding XML documents and return it to the user.


REDUCE

REDUCE Settings

Mandatory Qualifiers

|  |  |
|---|---|
| Gene Expression File | ta/Reduce/GeneExpression.txt |
| Organism | yeast |
| Motif Regular Expression | ???? |
| Upstream Sequence Length | 600 |
| Downstream Sequence Length | 0 |
| Max. number of motifs in result | 20 |

☐ Ignore coding regions

Results (Motifs by rank)

| ID | Motif | Significance | Chi Square |
|---|---|---|---|
| 1 | AAAA | 0.7022 | 0.5068 |
| 2 | TTTT | 0.6687 | 0.5527 |
| 3 | AAGA | 0.4213 | 0.8224 |
| 4 | AAAG | 0.4154 | 0.8274 |
| 5 | GAAA | 0.3898 | 0.8480 |
| 6 | AGAA | 0.3860 | 0.8509 |
| 7 | TCTT | 0.3741 | 0.8599 |
| 8 | TTTC | 0.3603 | 0.8701 |
| 9 | TTCT | 0.3540 | 0.8746 |
| 10 | CTTT | 0.3463 | 0.8800 |
| 11 | AAGT | 0.3234 | 0.8953 |
| 12 | ATTT | 0.3212 | 0.8968 |
| 13 | AAAT | 0.3185 | 0.8985 |

Gene: TFC3

Figure 5: Gene expressions.

We now give a simple example of a query using XomatiQ. The detailed discussion on the syntax and semantics of this query language is beyond the scope of this paper. Consider the warehoused EMBL database as introduced in Section 2.2.1. Suppose the user wants to lookup all records within the warehouse that have a certain attribute 'molecule' set to 'RNA', and have a reference to another database called SWISS-PROT [13]. This query can be expressed by the following XomatiQ query:

```
FOR      $a   IN
              document("hlx_embl_seq.hum")
              /hlx_n_sequence
WHERE         (($a//n_sequence_global
              _parameters/@molecule = "RNA" )
AND           ($a//db_reference/
              @db_code = "SWISS-PROT") )
RETURN        ($a//db_entry)
```

In the case of the gRNA, the document "hlx_embl_seq.hum" could be located on any database that is registered within in the same Data Grid (see Section 4.2).

Note that the query evaluation to correlate multiple warehouses in the gRNA is supported by efficient indexing mechanism. Details of the indexing mechanism are proprietary and therefore beyond the scope of this paper.

## 2.3 Genomics API

The Genomics APIs aim to provide domain specific programmable functionalities specific to the fields of genomics, molecular biology, cytology and genetics. However, the objective in still naming it the 'Genomics' API is to indicate the strong bent for *genomics-centric* way of research that the gRNA was designed for. The Genomics APIs contain object-oriented abstractions of biological *primitives*. This includes models of physical entities such as proteins and nucleotides; and abstract, functional entities such as 'genes'.

Typical attempts at creating such abstractions within the biological world tend to introduce the following biases – the nomenclature of elements within each primitive; the structural organization of information representing primitives; the inclusion of specific elements in association with specific primitives. For example, the scientific community may differ widely in terms of its definition of what a 'gene' is [29]. Some of these differences are trivial, and have to do with nomenclature of constituent terminology. However, even the inclusion of specific features within the sequence that should constitute a gene is under continuous debate and evaluation.

To minimize such bias, all classes within the Genomics API inherit from a basic class called the *BioObject* (see Figure 4). The BioObject does not contain the data structure of elements that should represent the particular primitive. Instead, it simply links to an external XML based DTD that stores the entire representation. It also links to an external XML document that stores information regarding the particular object instance of the class. All classes derived from BioObjects are accessible from a centralized directory service called the *Class Resource Manager*. By maintaining a class hierarchy using this format, the data structure of a primitive class can be extended by simply inserting elements within the existing DTD of the class, and then making the new class point to the new DTD. Moreover, this makes object-instances of the class conducive to easy storage within the database. All persistent members of a class are always maintained within the DTD. When we need to save the object-instance persistently, the entire XML object pointed to by the class is saved. Individual elements within the XML-representation are made available through simple accessor and mutator methods , that also impose strictly object-oriented methodology in accessing data mem-

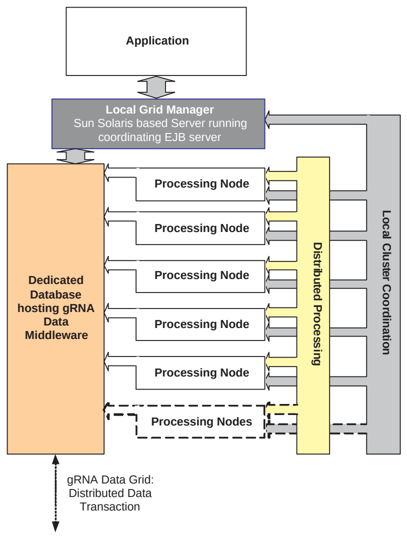

Application
Local Grid Manager
Sun Solaris based Server running coordinating EJB server
Dedicated Database hosting gRNA Data Middleware
Processing Node
Processing Node
Processing Node
Processing Node
Processing Node
Processing Nodes
Distributed Processing
Local Cluster Coordination
gRNA Data Grid: Distributed Data Transaction

Figure 6: gRNA Grid: Clustered Deployment Environment.

bers of classes; Alternately, one can choose to use the XomatiQ language to make queries within the document.

### 2.4 Other Components

The other programming interfaces include the Workflow and Visualization APIs, the Policy API, and the Systems and Grid API. Whereas parts of the Systems API, such as distributed programming, are explained in the next section, we will provide a brief overview of the remaining subsystems here.

The Visualization API provides access to a number of visual displays that would constitute useful interfaces in typical genomics-centric applications. This includes multi-modal interfaces for displaying large sequence data, interfaces for annotations and sequence analysis, displays for showing trees (such as phylogenetic trees) in various formats, displays for gene expression and microarray analysis, displays for showing XML data, displays for annotating XML tree.

The Workflow API provides an interpreted execution environment within the gRNA, and the programmable interfaces to create new classes that can run in this environment. This makes it possible to build an ad-hoc pipeline of tasks in a configurable manner. A generic drag-and-drop graphical interface provided to the end-user enables the visual construction of complex pipelines consisting of tasks, queries and data-sources. The Workflow API supports iterative and conditional operations, and performs design-time checking of new pipelines as they are constructed by the end-user.

## 3 Application Development Using gRNA Development Platform

To prove the feasibility of developing applications using the gRNA, we demonstrate the development of software around a technique from a publication titled *"Regulatory element detection using correlation with expression"* [21]. The paper presents a new computational method for discovering cis-regulatory elements that circumvents the need to cluster genes, based on their expression profiles. This is typically the sort of technique that a bioinformaticist might want to quickly experiment with, by applying it on his own data sources.

The development of the application (see Figure 5) involves the use of three gRNA APIs. The application requires data made available from databases from the European Molecular Lab (EMBL) [4] and data that must be specifically licensed from Stanford Genome Database (SGD). When we assume the task of developing this application, we must first contact Stanford to allow us to use the data in the manner that we intend to. In particular, we must seek permission to write a Data Hound to specifically access data from the SGD. Once the permission has been given, the developer uses the Data Hound API to create a mapping between the file-formats made available within the SGD, to a new internal DTDs. It must also use one of several internal transport protocols available within the Data Hound API to conduct the physical transport of the SGD data into a local warehouse. Once these mechanisms have been specified, the Data Hound can be deployed, and will ensure that the data is managed within the gRNA using the proprietary indexing mechanism within the data middleware.

Next, the developer can build a data service around the warehoused data from the SGD. This must include using one of the existing biological models available within the Genomics API as the basis for constructing the model. In particular, we take advantage of information regarding the physical positions of genes within the database and map related genome sequence information to a biological primitive type within the Genomics API called *DNASequence*, which represents DNA Sequences in general. Querying of specific genes within the local SGD warehouse involves the use of the XomatiQ API to conduct the queries. If we were to avoid this ideal form of abstraction, we could skip the step to write a data service class and directly employ the XomatiQ API to make queries.

In this instance, we need find whether the gene exists between these two positions in the chromosome. The query is designed to return '-1' if the gene does not exist between the positions, otherwise, it returns the gene closest to the end position. The objective is to find all the end position of the genes between the given positions, sorted by end position of genes in descending order. In this particular example, 'chromosome', ‘start’ and ‘end’ are variables.

```
FOR       $b  IN
              document("hlx_sgd.all")
              /hlx_sgd_entry
WHERE         $b//@chromosomeNumber
              = " + choromosome + "
AND           (($b//Beginning_Coordinate
              < " + end + ") )
AND           $b//Ending_Coordinate
              > " + start + ")
OR            ($b//Ending_Coordinate
              < " + end + "
AND           $b//Ending_Coordinate
              > " + start + "))
SORT BY       $b//Ending_Coordinate DESC
RETURN        ($b//Starting_Coordinate);
              ($b//Ending_Coordinate)
```

The development of the specialized Data Service however ensures that the functionalities of the SGD can be easily employed in new applications that need not understand the intricacies of the manner of organization of data within the database.

Finally, we use the *GeneExpressionAnalyzer* class within Genomics API to perform the actual analysis of gene expression data. The developer can either choose to start with raw images representing results obtained from a microarray experiment. The Genomics API contains classes that enable the preprocessing and processing of raw images to extract information for the anaylsis of gene expression. Alternately, he can directly analyze tabular forms of gene-expression data resulted from third party microarray image-processing applications.

## 4 Deployment within the gRNA

In this section, we discuss the second component of the gRNA architecture, i.e., the deployment platform.

We believe that the simplicity of the deployment environment makes for an important consideration in the successful adoption of new programmable systems.

The gRNA environment operates on a clustered computing environment consisting of multiple computers that communicate over Ethernet, that we refer to as the gRNA Grid (see Figure 6). The typical application written for the gRNA must therefore execute as a multiple tier application, with parts of it executing on the client computer, and parts executing on several server-side, distributed computers.

Applications operate from the client’s computer by communicating with the cluster through a single computer that hosts an Enterprise Java Bean (EJB) server as application server. This coordinating server then identifies one or more ‘processing nodes’, which are computers running a small footprint EJB server, to perform the task of primarily executing the server-side functionalities of the application.

The use of a single coordinating server is beneficial in several ways. The cluster can decide, at the point of initiation of the application, decide the number of computers that need to be assigned for that application, depending on availability and necessity. Moreover, the coordinating server also maintains directories of available resources discussed in the previous section, using an LDAP server. Because it essentially treats these processing nodes as computing resources, it can be configured to interface with heterogeneous hardware as well as specialized resources such as vector computers.

The gRNA has been primarily written using the Java programming language, with support for interoperability with Perl and Python scripting languages, which are fairly popular within the bioinformatics community. Where performance is a consideration, functionalities can be written in C or C++ and interfaced to the overall system using the Java Native Interface (JNI).

The deployment environment takes into account the following issues:

- Several procedures in bioinformatics can have tremendous time and space complexity. The gRNA addresses the limitations of localized computing resources via distributed computing, and provides a high level programming environment to encourage the adoption of this methodology.
- The deployment environment also provides a systematic and transparent mechanism in which to access warehouses and Data Services that are physically distributed across multiple clusters, in a framework referred to as the gRNA Data Grid.
- Classes developed for the gRNA should be able to bear functionality designated to execute on the client-side or on the server side. There should be a simplified process to inform the gRNA as to which functionalities should execute on which machine.

### 4.1 Distributed Computing within the gRNA

Distributed programming functionalities within the system API are designed to allow a single application to take advantage of multiple processors in the clustered environment. This is useful in the development of computationally intensive algorithms that are conducive to parallelization, which is quite common for several biological sequence analysis techniques [38]. The gRNA provides, through the System and Grid API, routines that make it possible to perform such parallelization using high-level interfaces similar to the threading capabilities provides in Java and POSIX. For the sake of simplicity, several of these functionalities are made compatible with the Message Passing Interface (MPI) and Parallel Virtual Machine (PVM) that are popular interfaces for distributed programming.

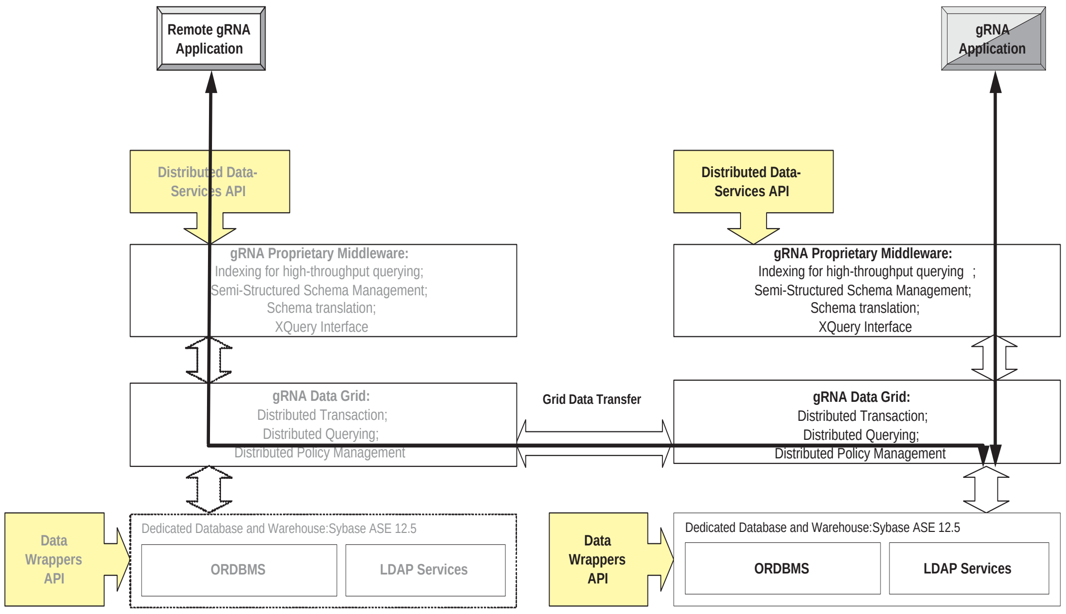

Remote gRNA Application
gRNA Application
Distributed Data-Services API
Distributed Data-Services API
gRNA Proprietary Middleware:
Indexing for high-throughput querying;
Semi-Structured Schema Management;
Schema translation;
XQuery Interface
gRNA Proprietary Middleware:
Indexing for high-throughput querying ;
Semi-Structured Schema Management;
Schema translation;
XQuery Interface
gRNA Data Grid:
Distributed Transaction;
Distributed Querying;
Distributed Policy Management
Grid Data Transfer
gRNA Data Grid:
Distributed Transaction;
Distributed Querying;
Distributed Policy Management
Data Wrappers API
Dedicated Database and Warehouse:Sybase ASE 12.5
ORDBMS
LDAP Services
Data Wrappers API
Dedicated Database and Warehouse:Sybase ASE 12.5
ORDBMS
LDAP Services

Figure 7: gRNA Data Grid Architecture.

When using these APIs, the cluster automatically imposes restrictions that prevent a developer from using arbitrary resources that have not been currently assigned to the current application.

### 4.2 The gRNA Data Grid

The gRNA Data Grid is a proprietary architecture to facilitate the management and querying of data warehouses located physically across multiple clusters. The Data Grid sits between the middleware and the actual database core (see Figure 7), making it transparent to the middleware as to the physical location of the data. The Data Grid uses a host of protocols to facilitate the transfer of data.

The Data Grid also allows for a potentially large scale federation of networked data databases. This is particularly beneficial in managing very large amounts of data, because the cost of managing multi-terabyte to petabyte sized databases can be prohibitively expensive, difficult and require the use of proprietary solutions.

As shown in Figure 7, the use of the Data Grid is made transparently within all data middleware APIs. This means that a developer writing a query correlating information between warehouses located on separate clusters, would write it as though the warehouses were on the same cluster. The gRNA middleware studies the query and then appropriately redirects parts of the query to databases where the related data is hosted.

In a multi-clustered environment, specific optimizations are made within the resource allocation schemes of the coordinating cluster that make it conducive to applications in genomics research. For example, because many genomics-centric applications frequently find themselves engaged in querying large data repositories, the system has the provision to assign applications to clusters that physically host their data of their interest.

### 4.3 Class Deployment

An important aspect of developing new classes in the gRNA is the ease with which functionalities within each classes can be designated to operate on the client-side or on the server side. We briefly cover how developers can create new distributed classes, how to build those classes, and how to create client class which can execute method remotely on them.

Figure 8 illustrates a simple case of third-party development in distributed gRNA system. Assume that the developer wants to write a class called *UseSequence* that extends gRNA *DNASequence* class, and subsequently wants to make calls to this class from client side class called *TestClass*. Classes (*UseSequence* and *TestClass*) in Figure 8 are the only classes written by the developer directly.

After passing through the source code through gRNA Source Code Generator, an additional six files are created. Three out of the six files are to be used by the cluster coordinating server, and are also needed to reside in the client side, since they hold the interfaces and helper functions for operating on the remote computer. The remaining three classes, which we refer to as *Node* classes are deployed in the nodes, but copies of them also resides in the cluster coordinating server as remote interfaces.

When client makes a function call, a 'proxy' object in the cluster coordinating server immediately knows

```
1 package com.fictitious.test;
2
3 import hlx.grna.biomodel.sequence._ ;
4 /**
5 *@ejb : bean type="Stateful"
6 *name="UseSequence "jndi_name="com/fictious/test/UseSequence"
7 *@ejb : transaction type="Required "
8 *@ejb : transaction_type type="Container"
9 *@jboss : container_configuration name="Standard Stateful SessionBean"
10 */
11 public class UseSequence
12 extends hlx.grna.biomodel.sequence.DNASequence
13 implements hlx.grna.system.Server.Object
14
15 public UseSequence( )
16 // default constructor
17 //add your code here . . .
18
19
20 //____________________________________________________________________
21 //add your methods here . . .
22 /**
23 *@ejb : i n t e r f a c e_method view_type=" remote "
24 */
25 public String callMe ( String in)
26 System.out.println( " I am called by this parameter : "+in ) ;
27 return "Thanks f o r the "+in ;
28
29
30 /**
31 *@ejb : interface_method view_type="remote "
32 *@hlx : returnType type="remote"
33 */
34 public RNASequencecreateSequence ( )
35 return new RNASequence ( ) ;
36
37
38 /**
39 *@ejb : interface_method view_type="remote"
40 *@hlx : returnType type="void"
41 */
42 public void doSomething( ) {
43 System.out.println ( "Do something " ) ;
44 }
45
46 /**
47 *@ejb : interface_method view_type="remote"
48 *@hlx : returnType type=" remoteArray"
49 */
50 public RNASequence [ ] createMultipleSequence ( ) {
51 RNASequence[ ] seqs = new RNASequence [ 2 ] ;
52 for ( int i =0; i<seqs.length; i++) {
53            seqs [ i ] = new RNASequence ( ) ;
54            }
55 return seqs;
56 }
57
58 // default methods, do not modify ! !
59 public void ejbActivate( ) {}
60 public void e jbPassivate ( ) {}
61 public void ejbRemove( ) {}
62 public void setSessionContext ( javax.e jb.SessionContext ctx ) {}
63
```

Figure 8: Sample code to demonstrate automated deployment of distributed classes.

which remote object to delegate this call to. The call is then delegated to the remote object at the node and executed there.

In order for classes to be able to run on gRNA distributed platform, some changes need to be done on the source code (see Figure 8). Line 4-10 are standard headers needed by the source code generator for generating the various proxy classes mentioned above. The *name* field at line 5 would be modified to reflect the name of the class, in this case is *UseSequence*. The *jndi-name* field at line 6 corresponds to the fully qualified class name (include package name) with '/' replacing the normal '.'. All gRNA remote object needs to implement a pre-defined *hlx.grna.system.ServerObject* interface. Alternatively they can extend one of gRNA distributed class, such as *DNASequence*. In this example the class implements *ServerObject* and extends *DNASequence*, even though extending *DNASequence* should suffice.

Line 15-18 are the default constructor. In creating a distributed class, we enforce that there should be one empty constructor. Additional constructor can be added as required. Functions in line 58-62 are standard methods and should not be removed nor changed by developer. The developer is then free to create any methods or variables normally as in normal Java classes. If the developer wanted to make some function remotely executable, he has to add line 23 to the function's javadoc comment.

The Object's variables are not directly available to the client at client side. The Genomics API also enforces this for reasons highlighted in Section 2.3. Access object's variables developer through accessor and mutator methods. Line 40 needs to be added if the function returns void. Line 32 is only added when the return value of the function is remote class. If this line is not added, the client gets the serialized version of the class, which cannot be used for remote execution. In case the developer does want the function to return the remote reference of distributed object instead, then line 32 must be added to the javadoc of the function. If the return value is an array of remote objects, the developer must add line 48 to the javadoc comment.

## 5 Related Work

There exist a number of systems that provide modular and configurable mechanisms for many of the issues in computational biology.

A number of notable systems target a key bottleneck within bioinformatics problem-solving, by providing systematic approaches to the issue of biological data integration and management. DiscoveryLink [7] from IBM follows the approach of providing configurable wrappers as consistent interfaces to the wide range of remote data sources. The Sequence Retrieval System (SRS) [43], initially developed at the European BioInformatics Institute and then further developed by Lion BioScience AG, provides a data management and query mechanism built around systematically warehousing databases as a federation. The Kleisli [20, 22, 42] system provides a systematic approach to managing and integrating external databases, and uses a functional query language to perform correlations across nested databases. Acedb [30] is an open-source special purpose object-based database designed to manage biological data.

There are a number of toolkits designed to encapsulate functionalities from specialized areas within computational biology. Notable examples include BioJava [1] and BioPerl [3], primarily designed for sequence analysis; and the Phylogenetic Analysis Library (PAL) [24]. The Ensembl initiative at the European Molecular Biology Labs (EMBL) [4], and the related Distributed Annotation System (DAS) [26], are systems that provide systemic, extensible approaches to the issue of annotating genomic data.

The gRNA distinguishes itself by factoring in the whole broad range of foundational requirements that go into typical applications in computational biology. It emphasizes on providing decoupled, yet inter-related subsystems that are essentially designed with the ease of third-party programmability as a key criteria. The gRNA also emphasizes on the ability to easily deploy new applications, and provides that as part of an integrative development and deployment environment.

## 6 Conclusions and Future Work

We have demonstrated how the gRNA provides an efficient and systematic mechanism for the development of new genomics-centric applications. In particular, we have demonstrated the use of a number of specialized programmable interfaces for the systematic management and integration of biological data. The gRNA also represents a successful strategy for consistently using XML for representing and querying biological data. By providing capabilities for the imposition of detailed policies on biological repositories, and by providing practical ways of dealing with the general lack of standards in biological data representation, the gRNA provides a practical backbone and an engineering approach to guide the development of genomics centric tools, as well as large-scale projects. We see the gRNA as a clear way to extending the 'hypothesis-based' way of general lifesciences research to the realm of bioinformatics research.

### Acknowledgment

We would like to thank members of the research and development team at HeliXense for making the gRNA a reality. This includes Sandeep Prakash, Arun Puri and Pathik Gupta for implementing the data related functionalities; Joshy George, Vivek Vedagiri and Wang Yi for their work on the bioinformatics-specific functionalities; Yusdi Santoso and Anand Sairam for their work on the Systems and Grid API; Sanju Sunny for his work on the workflow and visualization APIs; Joyce John for overall management of the project; Soh Fu Ling and Fenny Isanto for analyzing the intricacies of a large number of databases of interest.

## References

[1] The BioJava Project. www.biojava.org.

[2] The BioCORBA Project. www.biocorba.org.

[3] The BioPerl Project. www.bioPerl.org.

[4] European Molecular Biology Lab Nucleotide Database. http://www.ebi.ac.uk/embl/.

[5] Helixense Pte Ltd. www.helixense.com.

[6] Interoperable Informatics Infrastructure Consortium (I3C). www.i3c.org.

[7] IBM Life Sciences Discovery Link www3.ibm.com/solutions/lifesciences/discovery.html.

[8] Matlab. The MathWorks Inc.. www.mathworks.com/products/prodoverview.shtml, 1994-2002.

[9] Maya. Alias Wavefront. www.aliaswavefront.com/en/WhatWeDo/maya/see/solutions/includes/maya4brochure.pdf.

[10] Medline. http://www4.ncbi.nlm.nih.gov/PubMed/.

[11] National Center for Biotechnology Information's Genbank. http://www.ncbi.nlm.nih.gov/Genbank/.

[12] The Protein Data Bank. http://www.rcsb.org/pdb/.

[13] The SWISS-PROT Protein Knowledge Base. http://www.expasy.org/sprot/.

[14] The Systems Biology Workbench Project. www.cds.caltech.edu/erato/the_project.html.

[15] XQuery 1.0: An XML Query Language. www.w3.org/TR/2001/WD-xquery-20011220.

[16] T. K. Attwood, M. Bates, L. Bendekgey et al. Managing IPR in a Knowledge-based Economy - Bioinformatics and the Influence of Public Policy. *Report for the European Commission Research Directorate Generral*, Brussels, Belgium, 2001.

[17] R. B. Altman, T. Klein. Challenges for Biomedical Informatics and Pharmacogenomics. *Annual Rev Pharmacol Toxicol* 42:113-133, 2002.

[18] D. E. Bassett, M. B. Eisen, M. S. Boguski. Gene Expression Informatics – It's All in Your Mine. *Nature Genetics*, Vol 21, pp. 51–55, 1999.

[19] A. D. Baxevanis. The Molecular Biology Database Collection: 2002 Updates. *Nucleic Acids Res.*, Vol 30, pp. 1–12, 2002.

[20] P. Buneman, S. B. Davidson, K. Hart, G. Overton, L. Wong. A Data Transformation System for Biological Data Sources. *Proceedings of the 21st International Conference on Very Large Databases (VLDB 1995)*, 1995.

[21] H. J. Bussemaker, H. Lil, E. D. Siggial. Regulatory Element Detection Using Correlation with Expression *Nature Genetics,* Volume 27, pp. 167–171, 2001.

[22] S. Y. Chung, L. Wong. Kleisli: A New Tool for Data Integration in Biology. *Trends Biotechnol.*, 17(9), pp. 351–355, 1999.

[23] K. Decker, S. Khan, C. Schmidt, D. Michaud. Extending a Multi-Agent System for Genomic Annotation. *Cooperative Information Agents* pp 106-117, 2001.

[24] A. Drummond, K. Strimmer. PAL: An Object-oriented Programming Library for Molecular Evolution and Phylogenetics. *BioInformatics* 17 pp 662-663, 2001.

[25] S. S. Dwight, M. A. Harris, K. Dolinski, et al. *Saccharomyces* Genome Database (SGD) provides Secondary Gene Annotation Using the Gene Ontology (GO). *Nucleic Acids Res.*, Vol 30, pp. 69–72, 2002.

[26] R. D. Dowell, R. M. Jokerst, A. Day et al. The Distributed Annotation System. *BMC BioInformatics* 2:7, 2001.

[27] Data Retrieval Tasks Labeled as Impossible by the Department of Energy (1993), *DOE Informatics Summit Meeting Report*. Available via `www.gdb.org`.

[28] A. Hamosh, A. F. Scott, J. Amberger et al. Online Mendelian Inheritance in Man (OMIM), a Knowledgebase of Human Genes and Genetic Disorders. *Nucleic Acids Res.*, Vol 30 (1), 2002.

[29] H. F. Judson. Talking about the Genome. *Nature*, 15 Feb. 2001.

[30] S. Kelley. Getting Started with AceDB. *Briefings in BioInformatics* 1:2 pp 131-137, 2000.

[31] Q. Li, B. Moon. Indexing and Querying XML Data for Regular Path Expressions. *Proceedings of the 27th International Conference on Very Large Databases (VLDB 2001)*, pp. 361-370, Roma, Italy, 2001.

[32] A. Marshall. Laying the Foundations for Personalized Medicines. *Nature Biotech.*, 16 Sup 6–8, 1998.

[33] C. A. Ouzounis, P. D. Karp. The Past, Present and Future of Genome-wide Re-annotation. *Genome Biology*, 3(2), 2002.

[34] T. Reichhardt. It's Sink or Swim as a Tidal Wave of Data Approaches. *Nature*, 399(6736):517–520, 1999.

[35] G. M. Rubin, M. D. Yandell, J. R. Wortman et al. Comparative Genomics of the Eukaryotes. *Science* , Vol 287, 2000.

[36] R. Stevens, C. Goble, P. Baker, A. Brass. A Classification of Tasks in Bioinformatics. *Bioinformatics*, 17(2):180-8, 2001.

[37] J. Shanmugasundaram, K. Tufte, C. Zhang, et al. Relational Databases for Querying XML Documents: Limitations and Opportunities. *Proceedings of the 25th International Conference on Very Large Databases (VLDB 1999)*, pp. 302-314, Edinburgh, Scotland, 1999.

[38] T. F. Smith, M. S. Waterman. Identification of Common Molecular Subsequences. *Journal of Molecular Biology*, 147(1):195–7, 1981.

[39] M. E. van Eijk, L. F. Krist, J. Avorn et al. Do the Research Goal and the Databases Match? A Checklist for a Systematic Approach. *Health Policy* , 58(3):263–274, 2001.

[40] Y. Wang, J. B. Anderson, J. Chen et al. MMDB: Entrez's 3D-structure Database. *Nucleic Acids Res.*, Vol 30 (1), pp. 249–252, 2002.

[41] P. Wickware. Training on a Hybrid Discipline. *Nature Jobs*, October 2001.

[42] L. Wong. Kleisli, its Exchange Format, Supporting Tools, and an Application in Protein Interaction Extraction. *BIBE 2000*, pp. 21–28, 2000.

[43] E. M. Zdobnov, R. Lopez, R. Apweiler, T. Etzold. The EBI SRS server-recent Developments. *Bioinformatics*, 18(2), pp. 368–373, 2002.

[44] C. Zhang, J. F. Naughton, D. J. DeWitt et al. On Supporting Containment Queries in Relational Database Management Systems. *Proceedings of SIGMOD* , Santa Barbara, USA, 2001.