In Silico Biology 2, 0005 (2002); ©2002, Bioinformation Systems e.V.

Dagstuhl Seminar "Functional Genomics"

BioDataServer: A SQL-based service for the online integration of life science data

Andreas Freier[1], Ralf Hofestädt[1], Matthias Lange[2], Uwe Scholz[2], Andreas Stephanik[2]

[1]AG Bioinformatics / Medical Informatics, Technical Faculty,
University Bielefeld,
P.O. Box 100131, 33501 Bielefeld

[2]AG Bioinformatics / Medical Informatics, Institute for Technical and Business Information Systems,
Otto-von-Guericke University Magdeburg,
P.O. Box 4120, 39016 Magdeburg

Edited by E. Wingender; received November 01, 2001; revised and accepted December 29, 2001; published January 23, 2002

Abstract

Regarding molecular biology, we see an exponential growth of data and knowledge. Among others, this fa reflected in more than 300 molecular databases which are readily available on the Internet. The usage of these data requires integration tools capable of complex information fusion processes. This paper will present a novel concept for user specific integration of life science data. Our approach is based on a mediator architecture in conjunction with freely adjustable data schemes. The implemented prototype is calle BioDataServer and can be accessed on the Internet: http://integration.genophen.de. To realize a comfortable usage of the resulted data sets of the integration process, a SQL-based query language and a XML data format were developed and implemented.

Key words: data integration, data fusion, mediator, wrapper

Introduction

Today, more than 300 Molecular Database Systems (MDS) are available via the Internet [1]. Most of these database systems are developed and implemented by experimental scientists. All available data is directly indirectly retrieved from the lab. It is retrieved directly in the case of in-house database systems of experimental groups and indirectly in the case of literature and/or database search. Nevertheless, the u information is selected by experts who study papers, experiments and/or databases. With regard to the databases, this is more or less the fact. For this, powerful MDS for genes, proteins, transcription factor biochemical pathways, signal pathways and so forth are available (see http://www-bm.cs.uni-magdeburg.de/iti_bm/marg/dataacquisition/data_sources.html). Consequently, the Internet turned into a very, if not the most, popular medium for information retrieval.

Using MDS it is the current situation that we will find further information by explicitly cross linked data sources using hyper links. This situation requires the user to spend a lot of time for intelligent surfing. However, one of the the main problems of the manual Internet navigation is that linked URLs are outdated and the interesting data is distributed over several pages. A user friendly tool for the automatic selection and retrieval of user specific data is, thus, an important research topic for molecular biology.

The integration of different data sources into one user friendly interface requires a number of steps to implemented. Firstly, extracting data from the special MDS. Secondly, combining different data extraction analyzing processes [2], which is called the information fusion process. For example, the widespread utility BLAST [3] for the identification of gene similarity can not be implemented without the access to a pool of genomic databases. Therefore, tools for the integration of databases systems and the implementation of a user friendly query language will allow automatic data access. This again will support the information fusion process which can help to produce new hypotheses. The implementation of these tools is one task of bioinformatics. There are a number of systems described in the standard literature. We will review these in the next section.

We must consider a standardized, integrated, and homogeneous access to valuable databases in order to take advantage of their common content. Such a standardized solution should be supported by a powerful que language like SQL to access data in an homogeneous fashion. Thus, the integration of databases and the offering of a declarative query language can help scientists to detect new information and coherence to correlations across the spectrum from genomics, proteomics up to drug design. The idea of our approach implement a user specific integration tool based on a SQL enabled mediator system combined with wra (adapters) which are developed by semi-automatic software tools.

In this paper we will present an overview of our mediator based solution for the integrated retrieval of molecular biology data. We will continue to develop this idea into our BioDataServer (BDS) concept. We will use a tr synthetic metabolic scenario of enzymes and biochemical reactions, as shown in Fig. 1, to illustrate its data retrieval and query capabilities. Our example will use KEGG [4], BRENDA [5] and MDDB [6] as data sources.

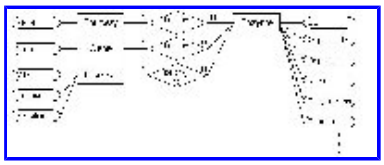

Figure 1: ER-Model of a synthetic metabolic scenario.

We continue with a presentation of our approach for a semi-automatic generation of data source adapter and a feasibility study for real Bioinformatic applications using the BDS as a data import component. This will b concluded with a section on sample applications that show examples of possible uses of our prototype by S clients.

State of the art

The idea of the molecular data source integration is not a new one. In the beginning of the nineties, firs approaches were described in the work of P. Karp [7]. At the same time the requirements for these integration approaches were formulated in different publications such as [8].

For the analysis of existing approaches the definition of various distinguishing marks is essential. The *degree* of integration and the *materialization* of the integration results are suggested. The degree of integration was described as *tight* or *loose*. A system is called tightly coupled if all schemes of the integrated data sources are transformed into one common data model and a global scheme was created. A solution was described as loosely integrated, if a transfer into common data model was made, but no global scheme had been created. The materialization distingui materialized and view-based solutions. A materialized approach transfers information of all component dat sources physically into one new MDS. A view-based system generates logical views onto the original information of the integrated MDS.

Furthermore, it is necessary to investigate some interface properties. Those properties include the support of *query languages* (e.g. SQL or OQL), an application programming interface (*API*) or *standards* like JDBC, ODBC or CORBA and the support of *various output formats* of the integration results (e.g. ASCII-text, HTML or XML).

Advantages and disadvantages of different approaches can only be analyzed if comparable properties ar specified. Thus, the following five properties are proposed:

1. degree of integration,
2. materialization,
3. query languages,
4. API/standards and
5. various output formats.

SRS [9] is an abbreviation for Sequence Retrieval System. This approach is based on local copies of ea

integrated data source with a special format which is described in the Icarus language specification. Icarus can help representing the structure of the integrated data source. Through the use of these local copies, SRS completely *materialized*. But during this transfer into the new format no scheme integration is realized. Therefore, the *degree of integration* can be characterized as *loose*. SRS runs on a Web-server and is accessible via any Web-browser. An HTML-interface for data queries is provided. Furthermore, the system can be queried by constructing special URLs. But no *query languages* like SQL or OQL is supported. SRS offers also a C-API. *Various output formats* are possible (HTML or ACSII-text). One problem with the result presentation in SRS is the necessity to parse the outputs for a further computer-based processing. The absence of any scheme integration is a disadvantageous for the use of the SRS system.

Similar to SRS are the underlying methods and possibilities for the use of the Entrez system [10]. This system is provided by the American National Center for Biotechnology Information (NCBI). This system integrates only data sources of NCBI. Thus, *no materialization* of the integrated sources is realized. Entrez uses *views* onto the original sources. Consequently, scheme integration could not be established. Therefore, the *degree of integration* can be classified as *loose*. The statements about SRS in order to query the system are con transferable to Entrez. There are no standard *query languages*, no standardized *API*, or other interface standards like JDBC. HTML is the only interface provided. Another Entrez feature is the manual construction of special URLs. *Various output formats* prove to be useful. These include HTML or ASCII-text, as well, as XML and ASN.1 files. The greatest disadvantage of the Entrez approach is the restricted number of integrated data sources (only NCBI internal data sources) and the missing support of query languages. Contrarily, the various output formats, primarily XML, are advantageous for the use of this system.

Another integration approach, which is based on multi-database techniques is the TAMBIS system [11]. This system can be used through a Java applet. TAMBIS works with views in order to access its integrated dat sources. Due to the use of a multi-database query language, it is not necessary to built an integrated glob scheme. Therefore, the degree of integration can be described as loose. As a query language in TAMBIS, a kind of the Collection Query Language (CPL) [12] is implemented. CPL is hardwired into the system architecture. This is why it is not so easy to use this query language from outside of the system. Oth disadvantages of TAMBIS are the absence of an API, or other public interfaces. The number of input formats which is limited to one - generated by the Java applet, proves also to be disadvantageous.

We would like to continue the discussion with ISYS [13]. ISYS stands for Integrated SYStem and can be characterized as a component-based implementation. The main goal of ISYS is to provide a dynamic and flexible platform for integration of molecular biological data sources. This system is developed as a Jav application, which must be installed on a local computer. Different platforms like MS-Windows or Solaris supported. The locally installed system accesses the distributed data sources on the Internet. One main feature is the global view onto the integrated data sources with the help of a global scheme. Materialization of the integrated sources is not required. ISYS provides a JDBC-driver. This feature implies SQL as query language. We can note also that more than one output format of the integration can be generated. ISYS has many advantages, which were presented in the description above. However, the necessary local installation of the system can be described as a disadvantage.

We close our survey with a professional system called DiscoveryLink [14]. This system was developed by IBM. It is based on federated database techniques. A federated system requires the development of a global scheme. Thereby, the degree of integration must be rated as *tight*. DiscoveryLink accesses its original data sources through views. Read-only SQL is supported as query language. An JDBC and an ODBC-driver are also provided, and different output formats can be generated as well.

In summary, many integration approaches for molecular biological data sources are currently available. We have presented five representative and well-known systems. These systems are based on different data integration techniques, e.g. federated database systems (ISYS and DiscoveryLink), multi database syste (TAMBIS) and data warehouses (Entrez and SRS). The subsequently presented mediator based approac database integration is a useful supplementation for the existing ones. Thereby, we focus on a flexible and thin, but universal applicable solution with powerful query and retrieval capabilities.

A summary for the above considerations is give in Tab. 1.

Table 1: Properties of some presented approaches for database integration.

	degree of integration	**materialization**	**query languages**	**API**	**output forma**

SRS	loose	materialized	-	C - API	HTML, ASCII
Entrez	loose	views	-	-	HTML, ASCII,
TAMBIS	loose	views	CPL	-	-
ISYS	tight	views	SQL	JDBC	various
DiscoveryLink	tight	views	SQL	JDBC, ODBC	various

Mediator-based architecture

MDS are represented as heterogeneous read-only data sources for scientists. These systems are mostl accessible through HTML browsers requiring manual navigation. As mentioned before, more sophisticate interfaces and query languages are required for the automatic access by computer programs. This scenario and the problem-specific integration of heterogeneous data sources must respectively tolerate differences in:

- Data models, data formats and structures,
- Remote query and access possibilities,
- Data storage and indexing techniques.

Consequently, the basic requirements for data integration in life science are:

- the opportunity for a combination of as many as possible MDS,
- homogeneous access to original data,
- the implementation of data caching for an efficient data retrieval,
- the provision of flexible and problem specific data schemes,
- a high level query interface and
- standard application programming interfaces (APIs).

They must be considered sufficiently. Computational methods of bioinformatics as well as individual scientists must be able to access interconnected life science data. Therefore, one can summarize that a generally applicable approach for database integration in life sciences must meet the following properties: tight integration and views, standard query languages and APIs, and data delivery in common supported data formats. We can see in Tab. 1 that the discussed systems realize only portions of these properties.

The *mediator-wrapper* concept, introduced by Wiederhold in 1992, and well described by Öszu and Valduriez [15], should be the basis for a homogenized, integrative and efficient retrieval of molecular biology data in opinion. A *wrapper* exports some information about its source scheme, data and query processing capabilities for each data source. A *mediator* centralizes the information provided by the wrappers in a unified view of all available data (stored in a global data scheme), decomposes the user query into smaller sub-queries (executable by the wrappers), and fina gathers the partial results and computes the answer to the user query. We will use the term *adapter* because it will be demonstrated that not all properties of a mediator are necessary in our approach. The idea o mediator-based database integration has resulted in the system architecture of the *BioDataServer* (BDS) shown in Fig. 2.

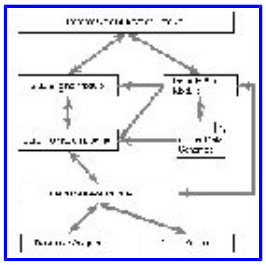

Figure 2: Architecture of the BioDataServer mediator-based database integration.

The proposed architecture is realized as a client-server system where the BDS is the server and a bioinformatics application can act as a client. Hereby, analysis tools are fully independent from the data retrieval components particularly with regard to the computational load, implementation details, computer platforms or location. The client platform is connected to the Internet preconditionally. A client application n connect as an authenticated user to the system and use its own profile for data retrieval on the basis of re individual integrated data schemes. In the following paragraph, a short overview of the server components is given.

The **Internet Communication Module** enables client applications access using the *Transmission Control Protocol/ Internet Protocol* (TCP/IP) for data transmission to and from the BDS. It should be possible to remotely control the BDS Mechanisms like user and process management, editing of global data schemes, data source wrapper control and integration progress information are provided by the **Remote Admin Module**. A high level data retrieval mechanism for a read-only access to the integrated data, with a high degree of system independence acceptance, is offered by the **SQL Engine Module**. Furthermore, the declarative character (specifying the properties of data to be retrieved and not how to obtain it) of the used mediator technique was taken into account. Therefore, a data retrieval subset of the well standardized *SQL* (*Structured Query Language*) has been implemented, which is one of the most popular query languages for relational database managemen systems (DBMS). The **Data Integration Module** is the core of the *BDS*, which includes a query and operator processor. In addition, the several global data schemes, which are again the basis for the integration process, are managed by this module. This integration process implements the *query decomposition* into sub-queries, the *transformation into integration operators*, *generation of execution hierarchy* and finally the *merging of sub-query results* in exactly that order. The **Retrieval Module** organizes all adapters in a similar way to an operating system which manages and controls the adapters. It *loads each single adapter*, *manages an adapter list*, *propagates the exported data source schemes*, *dispatches data source operations* and *ensures the robust adapter operation* (exception handling). The functions of a **Database Adapter** realizes homogeneous access to the data sources. For each wrapped data source the following tasks are performed: *reproduce a relational view* at the specific data model, *export a view* to the data source scheme and implement *data retrieval operations*
(see next section). Moreover, data intensive data retrieval tasks performed by the Database Adapter should be limited for the following reasons: avoid high workload on the data sources, reduce the net traffic and improve the system performance by minimizing the data extraction effort. Therefore, the **Cache Adapter** temporarily stores the previously retrieved data in an efficient manner (e.g. using a DBMS) and later queries are delegated to this cached data. Hence, a related update strategy is applied.

Database integration by integration operators

The described integration process must map the high-level SQL language, used for querying the BDS, to th diverse and often very limited query capabilities of the data sources. These are in turn closely connected to the used system (see Tab. 1). The remote accessible systems range from flatfiles to database systems and Web-sites. In consequence of the very different query facilities, it becomes necessary to find a commoı mechanism for comprehensive database system query processing. Therefore, the remote databases access and integration process is reduced to a common set of simple operations that are executable by every MDS These operations are split into two groups, which are defined on the basis of relational algebra operatioı introduced by [16] and are well described in [17]. It is assumed that the mathematical set theory is known. Thus we use mathematical set concepts and operations like *set*, *tuple* and *union*. On this basis, a *relation* $R(A_1, A_2, ..., A_n)$ can be defined informally as a set of tuples. Hence the tuple does not have any order. To simplify matters v assume that the data type of each tuple element (attribute) is a character string on which the *comparison operators*
(=,>,<,!=) can be applied. The problem of data type conversion is considered in the implementation of the BDS, but will not be addressed in this paper. Thus, it is possible to informally define the following operations on R:

Basic Operations

sel:

> This operation selects a subset of the tuples from a relation that satisfies a *selection condition*. The select condition is a Boolean expression specified on the attributes of R. The result from *sel* has the same attributes as R:
>
> $sel_{<\text{selection condition}>}(R)$

A selection condition is defined by a clause of this form:

<attribute> <comparison operator> <constant value> or,
<attribute> <comparison operator> <attribute>

proj:

This operation performs a projection of certain attributes from R and discards the other attributes. The result from *proj* is a relation including only the attributes specified in <attribute list>:

$proj_{<\text{attribute list}>}(R)$

cross:

This operation is used to combine tuples from two relations $R_A(A_1, A_2, ..., A_n)$ and $R_B(B_1, B_2, ..., B_m)$ in a combinatorial fashion. The result is a relation $R(A_1, A_2, ..., A_n, B_1, B_2, ..., B_m)$ with $n+m$ attributes. Furthermore, R has $n*m$ tuples - one tuple for each combination of tuples out of R_A and R_B:
$R \leftarrow R_A cross R_B$

For the homogeneous retrieval of data from the data sources we assume that each adapter provides *data source operations* which are based on a relation model. Consequently, each adapter must map the specific data model into relational one and the specific interface must map to the operators described below. Furthermore, we assume that each mapped relational view has a number of relations $SR(A_K, A_1, A_2, ..., A_n)$ which are in *third normal form* (3NF). That means that every nonprime attribute $A_1, A_2, ..., A_n$ of a relation must be only functionally dependent on the primary key A_K of the same relation. These concepts will not be further addressed in this paper, therefore the authors recommend for this the elaboration in [17].

Data Retrieval Operations

retrievepure:

This operator retrieves a relation of all primary keys out of a relation SR:

$retrievepure_{<A_r>}(SR) = proj_{A_r}(SR)$

retrieveselected:

This operator retrieves a relation including one nonprime attribute out of SR:

$retrieveselected_{<\text{attribute keyvalue}>}(SR) = proj_{attribute}(sel_{A_r = keyvalue}(SR))$

Query decomposition and data localization

In order to process a SQL query, mechanisms for query decomposition and data source localization must l applied. The required meta data concerning the structure of the global relations and the data localizatic respectively data distribution. These data are stored in so-called global data schemes, which are directly managed by the BDS. Using these meta data, a query decomposition algorithm maps a SQL query on a global relation into the previously defined **basic operations**. The data localization task takes the decomposed query as input and applies data distribution information and maps it to the **data retrieval operations**. Finally, a query plan based on a directed graph structure G = (N, E, S) is calculated, whereas each node N is an operator which handles a stream of tuples and the edges E are several instances of set operations. Additionally, S is function from E into ordered pairs of N.

The BDS is designed as a mediator. Consequently, there is not a single, global integrated data scheme b

collection of user specific global views called **user schemes**. Each user scheme consist of three basic concepts. The **data structure** is modeled in a relational way. Therefore, a hierarchy of *classes* (similar to a relation) including a set of *attributes* and related *data types* is used. Furthermore, for each modeled attribute, information about the **data sources** is gathered. Corresponding to the data retrieval operations, a set of *source attributes* can be specified as a tuple: <*adapter, relation, attribute*>. At least the **functional dependencies** between the source attributes must be specified since the construction of the execution plans is based on them. If we treat a class as a relation *VR*, a functional dependence between two source attributes *X* and *Y* out of *VR* is informally defined as a constraint, where the values of *Y* are determined by the values of *X* (denoted *X* $\rightarrow$ *Y*). Restrictively, no cycles are allowed.

The syntax of such a global scheme is given in the *Extended-Backus-Naur-Form* (EBNF):

<scheme> :== "scheme" <ID> {<class>}$^{+}$

<class> :== "class" <ID> "{\n" {<attribute>}$^{+}$ "}\n"

<attribute> :== <ID> ":" <data type> {<data source>}$^{+}$ ";\n"

<data source> :== <attribute source> [<functional dependency>]

<attribute source> :== "<" <ID> "," <ID> "," <ID> ">"

<functional dependency> :== ":" <ID> "->" <ID>

<data type> :== "string" | "integer" | "float" | "boolean"

The specification of global data schemes is a non-trivial procedure, which must be done with respect to the mapped application scenario. A support may be for example semantic enriched database schemes by u ontologies, thesauri etc. [18]. For illustration, in Fig. 3 a scheme is given for the previously modeled metabolic scenario.

Figure 3: Global integrated data scheme for a metabolic scenario.

Now, it becomes possible to calculate a query plan for each class within a scheme including all attributes, whereby all functional independence attributes are the start nodes of a query plan. Thus, for each valid SQL query an execution plan can be calculated which can be treated as a directed path through a query plan. example, a user wants to formulate the following, in natural language formulated queries:

> *Give me the EC-numbers and enzyme names for all known enzymes.*
> *Which biochemical reaction is catalyzed by the enzyme with the EC-number 3.5.3.1?*

In the second step, such natural language queries must be formulated in SQL. This could be done by a s predefined queries and the related mapping to SQL. In the case of software applications this task must be done by the software engineer of the graphical user interface. Furthermore, complex information systems ontologies [18] could be used. For example, this could result in the following SQL queries:

```
select ec, name from enzyme
select reaction from enzyme where ec = '3.5.3.1'.
```

The resulting decomposed queries are:

*proj*_{<ec,name>}(enzyme)
*proj*_{<reaction>}(*sel*_{<ec='3.5.3.1'>}(enzyme))

For the first query, the corresponding query plan can be calculated as shown in Fig. 4.

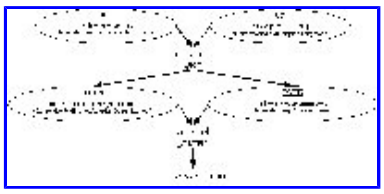

Figure 4: Bottom up query plan for retrieving EC-numbers and enzyme name.

To execute such query plans, an algorithm can be specified, which takes a SQL query and calculates an integrated relation (table) over all specified data sources, using the already defined integration operations. assume that a SQL query selects a set of attributes from a relation with several conditions. If more than one relation is used, joins must be explicitly performed. Because data retrieval operations are functionally limited) query conditions must be resolved by the mediator, too. The algorithm takes two parameters: a set of attributes *A*
(e.g. ec, name, reaction) and the name of the referred table in the used global scheme (e.g. enzyme). The scope of the algorithm is defined by the referred table. In Fig. 5, the resulting recursive algorithm on the basis of the integration operators that compute a relation *Result* is shown as pseudo code.

Figure 5: Algorithm for executing a query plan.

```
begin main
for each attribute ∈ A
    attribute result set RS_attribute = Ø
node = root_node of dependency tree
retrieve_node_data(node, Ø)
for each attribute ∈ A
    Result ← RS_attribute cross Result
end main

function retrieve_node_data( node , AV ):
begin function
    values = Ø
    attribute = node.attribute
    for each source attribute of node
        if AV = Ø
            values = retrievepure_sourceattribute
            RS_attribute ∪ values
        else
            for each val ∈ AV
                values ∪ retrieveselected_sourceattribute, val
            RS_attribute ∪ values
    for each son of node
        retrieve_node_data(son, values)
end function
```

Tool-supported adapter generation

As mentioned before, the BDS uses adapters to perform the described **data retrieval operations**. All adapters must implement a defined functional interface for the communication with the BDS. Thereby, the heterogeneous interfaces of the MDS must, of course, be reflected in the adapter. In a first systematizatic database management systems (DBMS), e.g. MDDB [6], and flatfile-based systems (data stream parsing), e.g. KEGG, can be distinguished. Because of the heavily heterogeneous character of the accessed MDS (Molecular Database Systems), the corresponding adapters must be programmed depending on their spe interface properties. Another requirement for each adapter is that it offers a relational view of the data which can be read from the data source. However, many data sources do not offer functions for the retrieval of scheme information in a relational form. Thus, the adapters must transform the original data model into a

relational one.

Therefore, semi-automatic approach for generating the adapters was developed, which provides an easy and fast opportunity for the adapter creation. Furthermore, a tool for the generation of adapters is implemented, which enables the access to database systems and flatfile systems. The tool needs specific information about the data access, a data scheme and in the case of file access a description of the files' structure. These specific information of a data source is saved in a text file, called *description file*. The description file will be processed by the generator and finally, the generated adapters can be connected to the BDS, which can map the data source operations to the data sources via the implemented function interface (see Fig. 6).

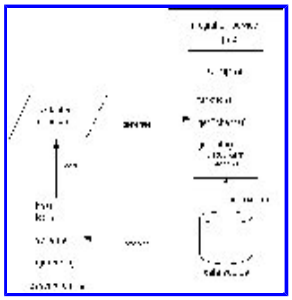

Figure 6: The process of adapter generation.

The first part of the description file represents information about network access parameters of a MDS (e.g. *URL*, *host*, *login*) and the second part includes scheme information. The scheme has a syntax similar to the data definition language (DDL) of SQL. In addition, a specification of the file structure of flatfile data sources must be provided in a third part of the description file. In the following paragraphs, the basic principles of DBMS and flatfile adapter generation will be exemplarily explained.

DBMSs in general offer methods for a client data access and methods to get the data scheme. Therefor information gathering for the adapter generation regarding the methods for access and the data scheme proves to be very easy. As most of the DBMSs provide drivers for standard APIs like ODBC or JDBC corresponding adapters can use this uniform interface for the database access. By that, the scheme of th database can be obtained, too, which can be adjusted within the description file. Because of the previous modeled MDS requirements and to show an example of a JDBC capable MDS, a part of the resulting description file for an access to our in-house database MDDB is shown in Fig. 7.

Figure 7: Extract of a description file for the MDDB.

```
adaptertype: JDBC
adaptername: GeneratedMDDBAdapter
manufacturer: "andreas stephanik"
version: 0.9a
jdbcdrivername: oracle.jdbc.driver.OracleDriver
host: jdbc:oracle:thin:@edradour.cs.uni-magdeburg.de:1521:
database: orcl
user: mddbguest
passwd: guestpw
scheme: mddb
create table AA_POS (
AA_POS_ID float,
NUC_AMIN_ID float,
AMINOACID_POS string,
AMINOACID string
)
create table ANNOTATION (
ANNOTATION_ID float,
ANNOTATION string
)
create table BIOCHEMICAL_REACTIONS (
ID_BC_REACTION float,
EC_NR string,
TYPE string
)
```

In the header of the file, the information of the host and login can be seen, after which the scheme of MDDB data can be found. This description file can be directly used as input for the adapter generator which generates the adapter for an access to the MDDB database. The access to a flatfile-based data source, which presents its data, e.g. in HTML or ASCII format, requires more manual effort because those data sources mostly do not offer methods to extract selected data directly or methods in order to obtain scheme information. Consequently, the required data must be extracted from the files by parsing. In the first step, the structure of the text files must be defined. Currently, *JavaCC* (http://www.webgain.com) grammars are used. A JavaCC grammar is defined in a separate file which is referenced at the end of the regarding description file. The JavaCC parser generator builds all necessary modules for parsing a specific text, e.g. HTML pages out of KEGG, which are included in the adapters. Fig. 8 shows an extract of the description file for the KEGG system.

Figure 8: Description file for KEGG.

```
adaptertype: USER_DEFINED_GRAMMAR
adaptername: GeneratedKeggAdapter
manufacturer: "Andreas Stephanik, mail: stephani@iti.cs.uni-magdeburg.de "
version: 1.0
url: http://www.genome.ad.jp/dbget-bin/www_bget?ligand+
keysurl: http://www.genome.ad.jp/dbget-bin/get_htext?ECtable+D
scheme: kegg
table Enzyme (ec_nr string primary key)
table Names (ec_nr string foreign key references Enzyme(ec_nr), name string);
table Classes (ec_nr string foreign key references Enzyme(ec_nr), eclass string);
table Sysnames (ec_nr string foreign key references Enzyme(ec_nr), sysname string);
table Reactions (ec_nr string foreign key references Enzyme(ec_nr), reaction string);
table Substrates (ec_nr string foreign key references Enzyme(ec_nr), substrate string
table Productes (ec_nr string foreign key references Enzyme(ec_nr), product string);
table Inhibitors (ec_nr string foreign key references Enzyme(ec_nr), inhibitor string
table Cofactors (ec_nr string foreign key references Enzyme(ec_nr), cofactor string);
table Comment (ec_nr string foreign key references Enzyme(ec_nr), comment string);
table Pathway (ec_nr string foreign key references Enzyme(ec_nr), pathway string);
table Gene (ec_nr string foreign key references Enzyme(ec_nr), gene string, genename
datagrammar: keggdatagrammar.txt
keygrammar: keggkeygrammar.txt
```

The URL of the KEGG system can be seen in the header of the description file. With respect to the data retrieval operations of the BDS, there are two URLs, one (called *keysurl*) is used in order to determine all key values. In the used example, all EC- numbers included in the KEGG system can be retrieved from the requested HTML page.

The second URL (called simply *url*) defines a pattern for building URLs to retrieve the data for an enzyme specified by its EC-number. For example, if a KEGG HTML-page including enzyme data for a given key shall be read, the related URL can be constructed by adding the key (EC-number) to the URL-pattern. The grammar for parsing the KEGG HTML-pages are referenced at the bottom in the description file. These are the references to files with the JavaCC grammars (*keygrammar*, *datagrammar*) which are used to extract all key values (from *keysurl*) or further data (from *url*). These grammars are used in order to describe the structure of files and extract the data. An example for such a grammar is given in Fig. 9.

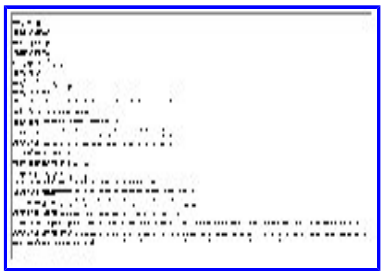

Figure 9: Extract of the grammar to parse and extract data from KEGG.

Moreover, it is possible to use other available options for the description and parsing of text files, e.g. *Perl* or *Python*
scripts. Therefore only the adapter type definition at the beginning of the corresponding description file must be adjusted (USER_DEFINED_PERL or USER_DEFINED_PYTHON) and the files with the scripts must be referenced at the bottom.

As mentioned before, a relational scheme of the data must be modeled manually. Since flatfile based systems do not offer a scheme explicitly, it must be modeled by the user, who also defines the description file. Finally, the adapter generator recognizes the type of adapter, creates the relevant components for parsing, transformation and the common interface for the communication with the BDS.

Application

As far as reported, the BioDataServer can be used to retrieve information provided by physically distributed data sources. Applications to the SQL interface are either interactive queries done by the user or algorith generated queries. In order to embed the BDS into existing software environments JDBC and ODBC drivers can be used.

In a reference example called phpMetatool (http://www-bm.cs.uni-magdeburg.de/phpMetatool) we applied MetaTool [19] to the BDS. MetaTool is a tool for the analysis of metabolic pathways. It is a classical C++ program that processes a proprietary flatfile format. One problem using MetaTool arises with the required manual provision of input data. A solution was to combine MetaTool with a direct access to integrated metabolic data and an interactive Web interface using the PHP language (http://www.php.net), instead of providing the input manually by editing a text file. To provide the integrated data to the phpMetatool, an integrated scheme for the BDS was modeled and transferred into an identical database scheme of an Or database. Furthermore, the data were imported into the database tables using the BDS. The database contains entries retrieved from BRENDA and KEGG. Applying this technique, the BDS was indirectly used for data acquisition. Now, the phpMetatool gives the user the possibility to interactively specify the data he wants to analyze (Fig. 10). Then, the selected data are converted into the flatfile format and analyzed by the MetaTool. Finally, the results are displayed as plain ASCII.

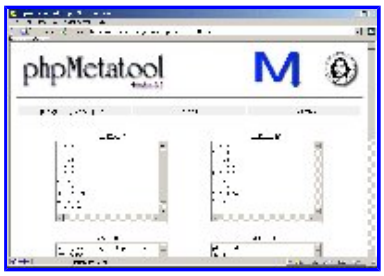

Figure 10: Screenshot of the phpMetatool.

Within the project "Modeling and Animation of Regulatory Gene Networks" (MARG) [20] (see http://www-bm.cs.uni-magdeburg.de/iti_bm/marg), the BDS was used to acquire data about gene regulative and metabolic networks from selected Internet databases. The idea of this project, which is supported by t German Research Council (DFG), is to present a virtual laboratory for the analysis of molecular process Therefore, it provides a fully scalable system for a user specific integration of different heterogeneous database systems and different interfaces to access the integrated data with special analysis tools (e.g. simulation environment that will not be addressed in this paper).

To handle the local storage of the fused data in a user-specific integration database, a component called IIUDB is used. By means of this *Individual Integrated User Database* [20] it is possible to build individual integrated databases that reflect the individual user's application requirements. Thus, it becomes possible to perform data analysis, cleaning, improvements, enrichment etc. The user is able to interactively specify and create th integration database in interface definition language (IDL) syntax. If the IDL is ready, the service modules will be generated automatically. The individual data import is based on specific integrated user schemes of the BDS that must be defined previously. The access to these integration databases is possible via the Commo Object Request Broker Architecture (CORBA).

The IIUDB provides the materialized layer for the data integration which is based on CORBA. The access importing integrated data from the BDS is possible using the JBDC driver of the integration layer. Furthermore, the IIUDB uses an integrated user scheme for the selection of attributes that should be integrated. An em database related to the integrated scheme is generated automatically. Once data from the integration service is read, it will be stored in the underlying standard object-oriented database. By storing the fusioned information in the cache, a separate new data source will be created. This new user specific integration database represents a quality of a meta database and is comparable to a data warehouse. The offered CORBA interface, similar to the BDS, enables other software tools to access the IIUDB.

Fig. 11 shows an example database scheme for the storage of gene regulation and metabolic processes. Th green-colored boxes in the class diagram are database classes, e.g. all enzymes are stored in the class "Enzyme". Grey-colored boxes define referential attributes that interconnect database classes with each other E.g., the reference "Reaction->ec" holds the information that an object of the class "Reaction" can be catalyzed by an object of the class "Enzyme". As a result, gene regulative networks are stored as homogeneous networks of database classes containing information from different heterogeneous databases.

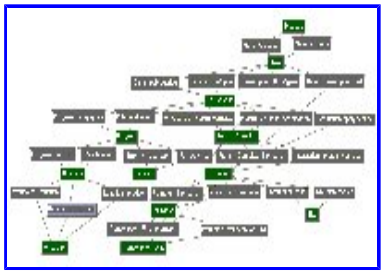

Figure 11: Example object network stored in the IIUDB.

For the design and the definition of the user specific databases, IDL is used. To support the design process, default IDL may be directly derived from a global data scheme out of the BDS. Mostly, the relational BDS attribute data types are strings. Therefore, the automatically derived IIUDB attributes will be strings, too. Ir modeling step between the derived scheme and the IIUDB configuration, the user can enrich the scheme v different data types, e.g. atomar/set attributes or literal/reference data types. Performing a data import, all pathway information will be automatically referred to the related enzymes. The example below shows a manually enriched IDL for an IIUDB database based on special parts of the introduced metabolic scenario. The simple IDL scheme schown in Fig. 12 includes four linked database classes as a fragment of the network of

Fig. 11.

Figure 12: IDL for a network of gene regulation and metabolic processes.

```
module metabolism {
// Pathway
interface Pathway {
attribute string name;
attribute sequence<Enzyme> ec;
};
// Enzyme
interface Enzyme {
attribute string ec;
attribute sequence<string> name;
attribute sequence<Reaction> reaction;
attribute sequence<Pathway> pathway;
};
// Reaction
interface Reaction {
attribute string name;
attribute sequence<Enzyme> ec;
attribute sequence<Metabolite> substrate;
attribute sequence<Metabolite> product;
};
// Metabolite
interface Metabolite {
attribute string name;
attribute sequence<Reaction> reaction;
};
};
```

To configure the system and in order to create the database, the IDL document must be processed with the **Configuration Tool**. After configuring the empty integration database, the IIUDB Browser enables the user control the data import. Two different import modes are available: **re-import** (all data will be imported) and **update**
(all database objects will be updated). Internally, a commercial object database system is used to store and retrieve the object networks. The IIUDB by itself is a separate CORBA service, wrapping the database obj together with the database functionality. Because all methods of the system are based abstractly on CORBA, we could relatively move easily to another database product, if it should be necessary.

Results and discussion

In order to take advantage of the potential of the valuable life science databases it has to be consider bioinformatics is an inherently integrative discipline, requiring access to data from a wide range of sources. Without the ability of a homogeneous access and moreover the combination of these data in new and interesting ways, the field of bioinformatics would be severely limited in scope. Consequently, existing approaches in the field of database integration in bioinformatics were analyzed, a nomenclature for the us technical solutions was introduced and finally a unique property catalog was suggested. On this basis representative systems were selected to investigate their individual advantages and disadvantages. Resulting from these considerations, a novel system for life science database integration was developed. The key idea of our approach is a mediator-based architecture in conjunction with semi-automatically generated adapters, which are developed within the MARG Bench project [20].

The BioDataServer has been designed to give a wide range of applications in bioinformatics the opportunity for a homogeneous access to non-materialized integrated life science data. With previously presented case studies of two existing bioinformatics applications, the practical applicability of this concept was demonstrat Hence, the whole power of the interconnected and analyzed life science knowledge can be used. The usable MDS are only limited by the number of currently implemented adapters. However, a technique for supporte adapter generation was presented that enables a fast (some minutes) generation of adapter for SQL enablec MDS. In the case of Web databases this job depends on the experience of the software developer and may

require few days till weeks. Using the provided SQL interface and the feature of the adapter cache as a time buffer for adjusting an adapter, it becomes possible to prevent the applications from changes in the MDS, due to logical data independence. A new MDS can be added using a new generated adapter and editing the related global data schemes.

A set of very simple operators which are used to calculate query plans were developed. These query plans are simple in such a manner that nearly all known MDS may be used to execute such a plan. For further work more complex operators may be considered dependent on the query capabilities of the particular MDS. In turn, this offers possibilities for query optimization.

The BioDataServer has been implemented as a JAVA application and provides remote services via a TCP/ socket. To ensure a wide acceptance and to give the possibility to evaluate the functionality of the BDS, necessary interface properties were realized. These include the support of a read-only subset of the SQL query language and the offer of a driver which implement the standard JDBC-API. Because this technolog enables universal data access for the Java programming language, any Java platform can be used to BDS-clients. Some template source code and the necessary driver are available using the Web-site: http://integration.genophen.de.

In addition to the simple JDBC access or ASCII-text generation, XML support is provided by a special drive add-on. XML is a standard format for flatfile data exchange and a various APIs for processing XML documents, which are implemented and freely available. Thus, it becomes possible to easily adapt existing analysis tools or complex information systems [21] to use integrated life science data retrieved by the BDS. Furthermore, this could be used for XML-based Web-publishing like the *Cocoon Framework* (http://xml.apache.org). A simple interactive Web-interface demonstrates these capabilities which is also available under the given URL. In addition, an ODBC driver for Microsoft Windows OS is currently under development.

First experiences with respect to BDS SQL queries were done by Java clients. The server platform used, was a Dual 450 MHz Intel Pentium system with RedHat Linux 6.0. The BDS-server application runs with a multithreaded Java 1.4 engine. Locally installed versions of KEGG and MDDB as well as BRENDA using remote SRS access to the EBI were used as data sources. The queries to the BDS were done by a Java clie using a 100MBit local network connection and the accessed data sources were available using the Internet. On the basis of the previously defined data scheme, the following SQL queries were processed on cached o uncached data, respectively. Tab. 2 shows the average times to retrieve the complete result set of the given query.

Table 2: Query performance tests of the BDS.

SQL-query	retrieved tupels	elapsed time (uncached)	elapsed time (cached)
select ec, name from gene	11579	4447 sec.	516 sec.
select ec, reaction from enzyme	3393	11807 sec.	632 sec.
select ec, name from pathway	2177	24 sec.	22 sec.
select ec, mim, disease, symptomes from disease	1362	217 sec.	173 sec.

Those measurements give an impression of the query behavior of the BDS. The latter is strongly dependent on

the respective parsing effort of the accessed MDS. The analysis of these results and the interpretation for further performance enhancements will be the object of further work. Thus, problems with respect to query plan optimization, e.g. by using the joint capabilities of the MDS, must be considered. In the case of the BDS, access through firewalls is not supported yet. A solution could be tunneling strategies using HTTP. Moreover, the implemented read-only SQL subset must be extended in order to provide a full support for the SQ specification.

Acknowledgement

This work is based on results of the *Research Group Information Fusion* and the project *Modeling and Animation of Regulatory Gene Networks*, which are kindly supported by the German Research Council (DFG).

References

1. Baxevanis, A. D. (2001). The Molecular Biology Database Collection: an updated compilation of biolo database resources. Nucleic Acids Res. **29**, 1-10.
2. Roos, D. S. (2001). Bioinformatics - Trying to Swim in a Sea of Data. Science **291**, 1260-1261.
3. Altschul, S. F., Gish, W., Miller, W., Meyers, E. W. and Lipman, D. J. (1990). Basic local alignment search tool. J. Mol. Biol. **215**, 403-410.
4. Kanehisa, M. and Goto, S. (2000). KEGG: Kyoto Encyclopedia of Genes and Genome. Nucleic Acids Res. **28**, 27-30.
5. Schomburg, D., Schomburg, L., Chang, A. and Bänsch, C. (1999). BRENDA the Information System Enzymes and metabolic Information, *In:* Proceedings of the German Conference on Bioinformatics (GCB '99), Germany, pp. 226-227.
6. Hofestädt, R., Mischke, U., Prüß, M. and Scholz, U. (1999). Metabolic drug pointing and informatio processing. Stud. Health Technol. Inform. **68**, 12-15.
7. Karp, P. D. (1995). A Strategy for Database Interoperation. J. Comput. Biol. **2**, 573-586.
8. Davidson, S. B., Overton, C. and Buneman, P. (1995). Challenges in Integrating Biological Data Sources. J. Comput. Biol. **2**, 557-572.
9. Etzold, T., Ulyanow, A. and Argos, P. (1996). SRS: Information Retrieval System for Molecular Biology Data Banks. Methods Enzymol. **266**, 114-128.
10. Tatusova, T. A., Karsch-Mizrachi L. and Ostell, J. A. (1999). Complete genomes in WWW Entrez representation and analysis. Bioinformatics **15**, 536-543.
11. Stevens, R., Baker, P., Bechhofer, S., Ng, G., Jacoby, A., Paton, N. W., Goble, C. A. and Brass, A. (2 TAMBIS: Transparent Access to Multible Bioinformatics Information Sources. Bioinformatics **16**, 184-185.
12. Davidson, S. B., Overton, C., Tannen V. and Wong, L. (1997). BioKleisli: a digital library for biome researchers. International Journal on Digital Libraries **1**, 36-53.
13. Siepel, A., Farmer, A., Tolopko, A., Zhuang, M., Mendes, P., Beavis, W. and Sobral, B. (2001). IS decentralized, component-based approach to the integration of heterogeneous bioinformatics re Bioinformatics **17**, 83-94.
14. Haas, L. M., Schwarz, P. M., Kodali, P., Kotlar, E., Rice, J. E. and Swope, W. C. (2001). DiscoveryLink: system for integrated access to life sciences data sources. IBM Systems Journal **40**, 489-511.
15. Özsu, M. T. and Valduriez, P. (1999). Principles of Distributed Database Systems, London et al.: Prentice-Hall, 2nd international edition.
16. Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. Communications of the ACM **13**, 377-387.

17. Elmasri, R. and Navathe, S. B. (2000). Fundamentals of Database Systems, Reading, Massachusetts et al.: Addison-Wesley, 3rd international edition.

18. Köhler, J., Lange, M., Hofestädt, R. and Schulze-Kremer, S. (2000). Logical and Semantic Databas Integration, *In:* BIBE 2000: IEEE International Symposium on Bio-Informatic & Biomedical Engineering, Washington, D.C., U.S.A.: IEEE Computer Society. pp. 77-80.

19. Pfeiffer, T., Sánches-Valdenebro, L., Nuño, J. C., Montero, F. and Schuster, S. (1999). METATOOL: f studying metabolic networks. Bioinformatics **15**, 251-257.

20. Freier, A., Hofestädt, R., Lange, M. and Scholz, U. (1999). MARGBench - An Approach for Integratio Modeling and Animation of Metabolic Networks, *In:* Proceedings of the German Conference on Bioinformatics (GCB '99), Hannover, Germany, pp. 190-194.

21. Hofestädt, R. and Scholz, U. (1998). Information Processing for the Analysis of Metabolic Pathways and Inborn Errors. BioSystems **47**, 91-102.