BioDataServer: An Applied Molecular Biological Data Integration Service*

S. Balko, M. Lange, R. Schnee, and U. Scholz

Institute for Plant Genetics and Crop Plant Research (IPK) Gatersleben, Germany
{balko|lange|schnee|scholz}@ipk-gatersleben.de

Abstract. Nowadays, huge volumes of molecular biological data are available from different biological research projects. This data often covers overlapping and complemental domains. For instance, the *Swiss-Prot* database merely contains protein sequences along with their annotations, whereas the *KEGG* database incorporates enzymes, metabolic pathways and genome data. Due to the fact that this data complements and completes each other, it is desirable to gain a global view on the integrated databases instead of browsing each single data source itself.
Unfortunately, most data sources are queried through proprietary interfaces with restricted access and typically support only a small set of simple query operations. Apart from minor exceptions, there is no common data model or presentation standard for the query results. Consequentially, the integration of manifold heterogeneous, distributed databases has become a typical, yet challenging task in bioinformatics. Within this paper, we introduce our own approach called "BioDataServer" which is a user-adaptable integration, storage, analysis and query service for molecular biological data targeted at commercial customers.

1 Introduction

Massive research activities in life sciences, i.e. biology in general and genome research, more specifically, have led to huge amounts of molecular biological data that originates from different projects. Due to the proliferation of the Internet, this data was made electronically available to a world-wide audience. Despite various integration efforts, in the majority of all cases, these data sources are either separated from one another (i.e. contain information on few specific subjects) or are loosely coupled (e.g. through hyperlinks). Some of these data sources became very popular which can be justified by (i) data quality, (ii) coverage of the research subject or simply by the pure (iii) volume of the data itself. For the time being, we incorporated five data sources into our integration scenario: *Swiss-Prot* [2], *TrEMBL*, *OMIM*[1] [6], *BRENDA* [12], and *KEGG* [9]. These protein related data sources are of particular interest for the research activities,

* This work was supported by the German Ministry of Education and Research (BMBF) under grant number 0310621.

[1] OMIM is a trademark of the Johns Hopkins University.

here at IPK Gatersleben and serve as a proof-of-concept for the *BioDataServer* integration concept.

Though most of these data sources are separated from one another, some of them provide a weak extent of integration with other data sources. For instance, an entry in the BRENDA database may incorporate homologous enzyme sequences imported from other databases like Swiss-Prot and TrEMBL. Other data sources prefer a reference-based coupling to further data sources. For example, the KEGG database hyperlinks to other data sources, like BRENDA.

Besides their non-integrated data domains, most of these data sources do not provide standard interfaces like JDBC or ODBC. In contrast, the data is queried through proprietary interfaces, mostly intended for human interaction (e.g. through web technologies like HTML forms, CGI-scripts, *Java Server Pages* and the like)[2]. Correspondingly, the possible query terms are of restricted expressive power and stay far behind a full-featured relational query language like SQL. Though, this restricted functionality may seem sufficient for some typical queries posed by a user, it inadequately supports more advanced applications with arbitrarily complex queries.

Even worse, query results to the web interface are mostly presented in a semi-structured manner. That is, different data sources do not obey common presentation guidelines but rather define structure and layout of the data output by themselves. At least, it exists an internationally acknowledged set of identifiers for some molecular biological data. For example, the EC number uniquely characterizes protein functions across different data sources and, thus, serves as a useful integration aid.

Finally, the data sources do not share a common (e.g. relational) data model. Therefore, many homogenization tasks are to be completed to obtain a properly integrated view onto the remote data sources. Our integration approach introduces the concept of so-called *attribute sets* to model the remote data schemas in a common way. Thus, attribute sets are the best compromise for a joint canonical data model and can basically be regarded as unnormalized relations. On the one hand, we are able to define common operations w.r.t. this data model. On the other hand, this plain data model poses minor requirements on the remote data sources. Some pre-integration tasks like schema homogenization, transformation to the relational model and partial normalization can automatically be performed by the integration service while other tasks still have to be conducted manually. Below, we enumerate the requirements to our integration service by the following characteristics from a database-specific point of view.

1. The motivation of our approach is driven by the objective of an applicable service to flexibly integrate various molecular biological data sources into a local customer database.
2. The integration service must establish relations between the data in the integrated relational data model. We emphasise complete information preservation rather than avoidance of redundancies.

[2] Recently, some approaches to XML data exchange have been proposed [1].

3. Our implementation must offer wide data analysis facilities and shall provide a high-level application interface which allows to formulate complex queries against the integrated data.
4. We must guarantee a fast, reliable access onto the integrated schema that is amenable to the addition of own data sources. In addition, customers must be given the opportunity to specify well-defined fractions of the remote data sources that are to be present in the integrated schema.

This paper is organized as follows. In Section 2 we survey competing integration approaches. In Section 3 we introduce our system's architecture and motivate its design concepts by the requirements for a commercially applicable implementation. Section 4 formally introduces our data integration approach and exemplarily explains its data extraction and merging strategies. In Section 5 we briefly portray two sample web applications that interact with the integrated data stock. Section 6 concludes this paper by means of an outlook on our ongoing research issues.

2 Integration Approaches

Currently, many integration approaches exist in the field of molecular biological data. [10] describes different integration approaches, besides simple hyperlink navigation. These are *federated databases* [13], *mediators* [17], *multi database queries* [10], and *data warehouses* [8]. Most of the existing systems can be classified as one of these four basic types of integration. Existing implementations can be compared by means of five basic properties which are: (1) integration approach, (2) degree of integration, (3) materialization of the integration results, (4) supported data types, and (5) expressive power of the query operators.

The *degree of integration* is described as being tight or loose. A system is tightly coupled if all schemas of the integrated data sources are transformed into one common data model and a global schema exists. Whereas, an implementation is loosely integrated, if a mapping into a common data model was conducted, but no global schema exists. The *materialization* distinguishes materialized and view-based solutions. A materialized approach physically transfers information of all participating data sources into one global database. In contrast, a view-based implementation generates logical views onto the integrated data. The analysis of *supported data types* distinguishes between atomic types like numbers for integer-valued identifiers or strings, and complex types (sets, lists, bags, etc.) for nucleotide and amino acids. The property *query operators* characterizes the expressive power of queries sent against the integrated data stock. For example, arbitrarily complex selection predicates can be considered powerful query operators. These also include non-standard comparison in pattern matching, which are heavily required in bioinformatics. Simple single-attribute exact match queries have less expressive power and do not match the requirements of many advanced applications.

Many useful integration approaches already exist. Among the most well-known approaches are SRS [3], the Entrez system [16], the TAMBIS system [15],

ISYS [14], and DiscoveryLink [5]. Table 1 classifies these systems according to the described properties. These systems are based on different data integration

	approach	integration degree	materia-lization	data types	query operators
SRS	data wareh. w/o global schema	loose	completely materialized	strings, NA & AA seq.	boolean pred., reg. exp., homology search
Entrez	data wareh. w/o global schema	loose	views (NCBI data sources)	strings, NA & AA seq.	boolean predicates homology search
TAMBIS	multi DB queries	loose	views	strings	case based qrs. ontologies
ISYS	federated DBMS	tight	views	strings	read only SQL
Disc. Link	federated DBMS	tight	views	strings	read only SQL

Table 1. Molecular Biological Integration Approaches

approaches, e.g. federated database systems (ISYS and DiscoveryLink), multi database systems (TAMBIS) and data warehouses (Entrez and SRS).

Our *BioDataServer* can be regarded as a (i) mediator-based approach that (ii) tightly couples the participating data sources in a (iii) completely materialized integrated data stock. Moreover, we support (iv) atomic types (string, integers, numbers) and partly sequence data (through BLAST coupling). Users can (v) modify the integrated data and query operations on the integrated data (vi) speed-up (no network load).

3 System Architecture

In short, our integration service must provide fast and reliable access to a customer-adaptable integrated database of molecular biological data. This data must be efficiently extractable from existing distributed, heterogeneous data sources ("remote data sources"). These condensed requirements bring forth some design decisions, we shall subsequently explain in detail.

Within the scope of database integration, one of the most important design issues is to either (i) physically integrate the remote data stocks into some materialized global database or (ii) to define a purely logical global view onto the remote data sources. Operating on physically integrated data guarantees a high availability with short response times and establishes an intermediate independence from schema changes in the remote data sources until the next update cycle. The most fundamental advantage, however, lies within the applicability of complex queries on the integrated data stock. That is, users can take advantage of a full-featured query language and its complete expressive power

which broadens the application domain towards data visualization programs, data-intense statistical analysis, data mining and so forth.

On the other hand, we face the situation to likely operate on old, possibly outdated information which can only be mildered by time-consuming frequent update cycles to synchronize the integrated database with the remote data sources. However, biologists often wish to access consistent data during their long-running experiments. That is, too frequent updates are often dispensable, anyway. In turn, a non-materialized global view onto the autonomous remote data sources avoids redundancies, does not require a large local data storage, and ensures up-to-date information at any time. However, this concept establishes a strong dependence on the remote data sources in many ways. To name only a few, constant availability, short response times, stable query interfaces and robust result format must be provided at any time. Moreover, a high transaction load on the integrated schema would cause high network traffic, ultimately leading to inacceptable response times.

Though the benefits of a non-materialized global schema may seem preferable in academic prototypes, the robustness criteria of a materialized integrated schema clearly outweighs its disadvantages in an industry-ready system. In particular, in case of alterations in the remote schemas, users applications can still operate on the integrated global schema while database maintainers have time to adjust the data source accession implementations (e.g. modify data source adapters) until the next update cycle. In the light of this substantiation, the disadvantages of a materialized global schema are tolerable.

Our second design decision lies within the tightness of the integration. In contrast to the remote data sources which partially provide references to other data sources and, thus, are loosely coupled on their own, our integrated schema merges the structural information of the remote data sources into a semantically integrated schema which abrogates the boundaries of the hitherto disjoint remote schemas. On the beneficial side, applications working on the global schema, are no longer bound to the structure of each remote data source but operate on a single uniform structure. Moreover, the integrated schema establishes semantic relations between the so far incoherent data and does not leave this difficult task to the application programmer.

Thirdly, customers typically wish to correlate the integrated data with their own data stock. That is, customer data must be merged into the integrated schema, as well. More specifically, most customers do not wish to upload their confidential research results onto a remote server to perform data analysis (e.g. sequence alignment). This decision is typically justified by security and nondisclosure issues. There are less security concerns when working on an own local database server.

Moreover, each customer has different requirements regarding the integrated data. An adaptable integration concept must be able to flexibly assemble a set of relevant data sources to be present in the integrated schema. We will introduce (i) industry standard interfaces and protocols to the communication paths between the integration layer and the adapters and (ii) a formalization of the basic data

extraction operations for automatic access plan generation which simplifies any adapter programming task to the very data source specific adaptations.

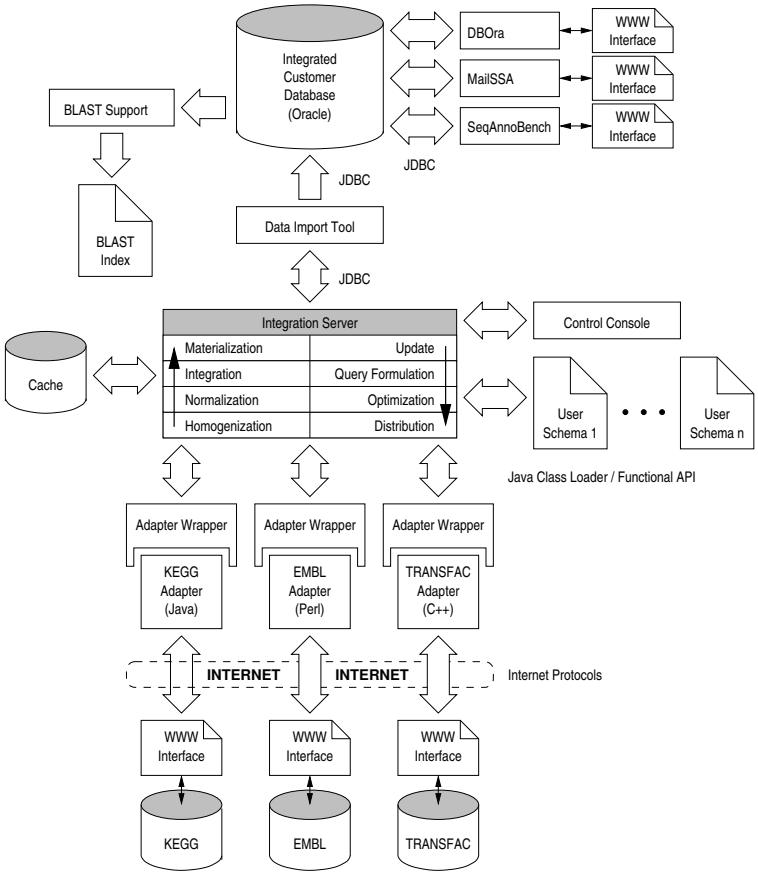

Fig. 1. System Architecture of *BioDataServer*

Our overall integration architecture is depicted in Figure 1. The remote data sources at the bottom level are distributed across the Internet. These data sources are mostly exclusively accessible via proprietary interfaces to external users. In some rare cases, remote data sources provide a relational access via industry-standard interfaces like JDBC on their own. Some other data sources can be duplicated, downloaded, and processed locally. However, an adapter usually accesses the remote data sources through a web interface and fetches its query results from a semi-structured HTML document. Whilst the query transmission can be conducted via HTTP protocols or URL assembly, gathering the result from the HTML page is merely a parsing task done by the adapter.

The integration layer runs as a server program which handles several users, each with its own integrated user schema, in parallel. That is, each user specifies its own integrated schema definition (cf. Section 4). Once, the integration layer receives an update request from a particular user, it initiates the update process which comprises (i) a formulation of data extraction requests to the remote data sources, (ii) an access plan generation and query optimization, and (iii) the distribution of the data extraction request with queries against the remote data sources.

When all data extraction requests are passed to their respective adapters, the integration layer waits for them to return their results. A cache stores intermediate results temporarily. When all adapters have finished their update tasks, the pre-integration process starts. That is, a homogenization resolves conflicting attribute names, and introduces common data types for *same*-attributes.

Altogether, the system architecture pursues a hybrid approach. On the one hand, customer applications interact with locally materialized data structured in an integrated schema. On the other hand, the integration service conceptually maintains an unrestricted number of user schemas (cf. Section 4) along with other metadata on data source capabilities and data extraction processing. This metadata is only being used to automate update requests of the integrated data as opposed to user queries that are posed against the locally materialized data stored in a relational DBMS.

4 Schema Integration

Each remote data source (RDS) may be structured in an own data model. Some data sources provide relational views onto their data while others present their query results as semi-structured text or HTML files. Therefore, prior to any data integration task those structural inconsistencies must be resolved by a transformation into a conceptual schema in our canonical data model.

In fact, the distinction between a *data source* and a *database* can be made by their architectural differences. A data source trivially is data which (1) may be distributed across several servers. Its content is (2) typically accessed through a joint web interface which only accepts proprietarily formulated queries. In contrast, a full-featured database manages data that are structured in a well-defined data model. Besides, it provides standardized query languages and interfaces like SQL and JDBC.

The main objective of the *BioDataServer* approach is to use the data in the RDS in a consistent, integrated manner. However, a loose, link-based coupling of the data sources does not serve this goal. To take full advantage of the potential of semantically related, yet physically distributed life science data, we must consider all participating RDS as one huge inter-connected information resource. State-of-the-art techniques present the RDS as inter-linked, yet isolated nodes of a global database network. Coupling rests on web technologies like HTML hyperlinks which satisfyingly support navigation but lack the ability to formulate arbitrarily complex queries like joins, grouping and aggregation, index exploita-

tion, or similarity queries. Nevertheless, novel applications heavily rely on these advanced querying facilities when operating on life science data to perform information retrieval, knowledge discovery, or data mining tasks. In consequence, a tight database integration which incorporates the RDS into a joint data inside a state-of-the-art DBMS will be enormously profitable. Thus an integration of structurally heterogeneous, distributed, yet semantically overlapping data must be provided by our integration approach. Due to the nature of distributed information resources, our integration mechanism must cope with a high degree of autonomy w.r.t. the RDS. That is, we must typically use the pre-defined (and somewhat limited) access paths of a RDS to "extract" its content. Obviously, operations on the RDS are restricted to read-only accesses. Other aspects of data distribution and heterogeneity which have to be considered by our integration approach are (i) data fragmentation and allocation, (ii) canonical data modeling, and (iii) schema mapping and homogenization. Those aspects are being reflected by the subsequentially described schema architecture of BioDataServer (BDS).

The BioDataServer comprises three schema levels. The *Remote Export Schema* (RES) resides at the remote data sources and provides the data schemas specified in their respective native data models. In other words, the RES is a structural description of a data source. The *Adapter Schema* (AS) is a view on the RES of a data source provided in our joint canonical data model. Finally, the *Integrated User Schema* (IUS) reflects a portion of the integrated adapter schemas of the participating data sources. The IUS follows our canonical data model as well.

4.1 Remote Export Schema

Despite the fact that the majority of the remote data sources where conceptually well modeled, their majority is not directly applicable to schema integration. For instance, they typically lack an explicitely specified export schema. The main reason for this disadvantageous characteristics lies within rigorous access restrictions to the backend DBMS of a data source which prevents explicit access to its schema specification and other metadata. More precisely, schema information can be provided as a DBMS catalog, a XML schema, an IDL or other standard formats for self-describing interfaces for databases.

In some cases, there may not even be an explicitely modeled schema which would follow some formal data model specification. Instead, the data source might provide informal schema information given, for example, as textual description in natural language. In consequence, we categorize the RES provision task into two tasks: *schema re-engineering* and *schema retrieval*.

As there is no explicitely provided export schema for most data sources, a re-engineering task must be conducted to create an RES from the actual data presentation (e.g. flat files) from scratch. This time-consuming process can be accelerated through the support of semiautomatic methods. For the time being, we generate export schemas and data extraction modules by a formal grammar-based approach [4]. Alternative projects like *WebJDBS*, *TSIMMIS*, *Garlic* or *W4F* as discussed in [7], are beyond the scope of this paper. Of course, all newly constructed RES share a joint data model which corresponds to the canonical

data model introduced in Section 4.2. That is, the export schema and the adapter schema match one another.

In some rare cases, the adapter can directly retrieve the catalog data of a data source through interfaces like JDBC, ODBC, SOAP/WSDL, and the like. As our implementation bases on Java, we favor the JDBC API to access data source catalogs by generated SQL statements and extract the export schema information automatically.

4.2 Adapter Schema

Adapter schemas were introduced to overcome the data model heterogeneity between export schemas. An adapter schema is specified in a joint canonical data model. The requirements for this canonical data model are twofold. On the one hand, it should support all important basic concepts that appear in a data source. On the other hand, it should be a semantically poor data model to ease the tasks of (i) schema mapping (RES to AS) and (ii) data integration (AS to IUS). Therefore, we introduce *attribute sets* as an abstraction of relations. That is, attribute sets assemble those attributes which are related, i.e. jointly presented by a data source. An attribute comprises (i) an attribute name, (ii) an attribute type, and (iii) meta information about the existence of access paths (*is indexed*) and functional dependencies (*is key*). Thus, it widely matches the concept of relations but additionally introduces retrieval functionality and lacks foreign keys.

Hence, the syntactical notation of an adapter schema can be formally defined as follows. We call $\mathbf{A}$ an attribute out of the set of possible attributes in a database schema $\mathbf{A} = (\mathbf{N}, \mathbf{D}, \mathbf{P})$, where $\mathbf{N}$ is the attribute name, $\mathbf{D}$ its atomic data type, and $\mathbf{P}$ a combination of "is (not) key" and "is (not) indexed".

The attributes are aggregated into an *attribute set* $\mathbf{R} = \{\mathbf{A}_1, \ldots, \mathbf{A}_n\}$. At the adapter level, the canonical data schema is complemented by so-called *retrieval dependencies*. In contrast to an SQL interface, retrieval from remote data sources is restricted to a set of pre-defined plain query operations. Typically, these query operations take the form (in SQL-like notation):

$$\mathsf{select}\ A_1, \ldots, A_i\ \mathsf{from}\ R\ \mathsf{where}\ B_1 = b_1\ \mathsf{and} \ldots \mathsf{and}\ B_j = b_j$$

Notice, that $A_1, \ldots, A_i$ and $B_1, \ldots, B_j$ are (typically disjoint) sets of attributes, and $b_1, \ldots, b_j$ is a list of attribute values that $B_1, \ldots, B_j$ must match. The concept of *subgoals* [11] expresses this basic retrieval functionality that any adapter supports. The notation is simplified to terms like $H(B_1, \ldots, B_j; A_1, \ldots, A_i)$ where $B_1, \ldots, B_j$ are so-called *input attributes* and $A_1, \ldots, A_i$ enlist the *output attributes*. In [11] the authors propose an algorithm to efficiently answer any data extraction request by a combination of subsequent subgoals. Unfortunately, their method ends up in solving a SAT problem by some heuristic approach. Therefore, we introduce a minor adaptation which restricts the number of input attributes to 1 or 0. Or, if no input attribute is provided, the adapter provides mechanisms to retrieve a list of all output attribute values.

Retrieval dependencies with more than one input attribute appear very rarely in existing data sources. Even if one might sometimes encounter these cases, there are typically many ways to circumvent them by sequels of simple $1:n$ retrieval dependencies. In consequence, we obtain a notable reduction of computational complexity for access plan generation (i.e. assembly of subsequently "executed" subgoals for a particular query operation), cf. Section 4.4.

In terms of a specific example, we consider the data sources KEGG and BRENDA. We abstain from an illustration of their native remote export schemas and specify their adapter schemas by means of our canonical data model. Here is a simple example: KEGG $= \{kr_1, kr_2\}$ comprises two attribute sets while BRENDA $= \{br_1\}$ contains one attribute set. The attribute sets kr_1, kr_2, and br_1 contain enzyme and pathway data:

$$\begin{aligned}
br_1 = \{&b_1 = (ec, \text{string}, \{\text{is indexed}, \text{is key}\}),\\
&b_2 = (name, \text{string}, \{\text{is not indexed}, \text{is no key}\}),\\
&b_3 = (reaction, \text{string}, \{\text{is not indexed}, \text{is no key}\}),\\
&b_4 = (organism, \text{string}, \{\text{is not indexed}, \text{is no key}\})\}\\
kr_1 = \{&a_1 = (ec, \text{string}, \{\text{is indexed}, \text{is key}\}),\\
&a_2 = (name, \text{string}, \{\text{is not indexed}, \text{is no key}\}),\\
&a_3 = (reaction, \text{string}, \{\text{is not indexed}, \text{is no key}\})\}\\
kr_2 = \{&a_4 = (map, \text{string}, \{\text{is indexed}, \text{is key}\}),\\
&a_5 = (pathwayname, \text{string}, \{\text{is not indexed}, \text{is no key}\}),\\
&a_1 = (ec, \text{string}, \{\text{is indexed}, \text{is key}\})\}
\end{aligned}$$

The attribute names *ec*, *name*, *reaction*, ..., which originate from the data sources, are replaced by the the attributes $a_1, a_2, \ldots$ (KEGG) and $b_1, b_2, \ldots$ (BRENDA) which appear in the AS. Notice, that some identifiers appear across many attribute sets. The retrieval dependencies in both data sources are given as follows:

$$\begin{array}{lll}
H_{\text{KEGG},1}(\,;a_1); & H_{\text{KEGG},2}(a_1;a_2,a_3); & H_{\text{KEGG},3}(a_1;a_4);\\
H_{\text{KEGG},4}(\,;a_4); & H_{\text{KEGG},5}(a_4;a_5,a_1); & H_{\text{BRENDA},1}(b_1;b_2,b_3,b_4);
\end{array}$$

That is, KEGG allows a retrieval of all stored EC numbers (a_1) and pathway maps (a_4). In contrast, BRENDA provides no facility to obtain a list of indexed attribute values and lacks some "entry point" for query processing.

4.3 Integrated User Schema

The IUS reflects our objective of a tight database integration where the integration component integrates the adapter schemas and maps the remote data into a materialized global database. Actually, several integrated user schemas may exist which reflect different "views" onto the integrated data. Each of these IUS covers specific data interests like enzyme and pathway information on a certain organism.

Hitherto, each adapter schema comprises its distinct set of attribute names. The IUS, however, needs to merge this information into an own set of attribute identifiers. That is, each attribute in the IUS is linked to one or more attributes in the integrated AS, i.e. their attribute values are supplied by the homogenized attribute values in the RDS. But how does the integration layer know which attributes in the participating RDS cover the same semantics? The magic is done by the concept of *same*-attributes which are manually specified by an $(n+1)$-ary relation in the IUS. More precisely, each *same*-relation links an attribute in the IUS with the attributes in the participating AS. In terms of our example, we model the following *same*-relations:

$$\mathsf{same}(ec, a_1, b_1); \quad \mathsf{same}(name, a_2, b_2);$$
$$\mathsf{same}(reaction, a_3, b_3); \quad \mathsf{same}(species, b_4); \quad \mathsf{same}(pathway, a_5);$$

Later on, we employ *same*-Attributes to merge attribute sets by means of join operations in the integration process. Unfortunately, molecular biological data items that express identical subjects often have different representations. This is particularly true for data that originates from different resources.

4.4 Query Processing

In the scope of this paper, the term *query processing* reflects a certain data extraction request sent against the integration layer to update the materialized data. Suppose, the user wants to update an attribute set in his integrated data. Queries against the integrated data are handled by its hosting RDBMS.

The integration layer processes a data extraction request in a series of steps, similar to ordinary database query processing. Currently, we have implemented a repository of static (i.e. manually designed) query plans to process those data extraction requests. However, progress has been made to (i) provide semi-automatism for a hitherto manual query plan design and, later on, to (ii) fully automatically process data extraction requests without any manual interaction which shall be described in this section.

First, the integration layer interpretes the query which is given in an SQL-like notation. For example, the user might request an update on an attribute set *Enzyme* which comprises the attributes *ec*, *name*, *reaction*, *species*, and *pathway*, for example: $\mathsf{select} \ * \ \mathsf{from} \ Enzyme$.

Next, the *same*-relations come into play to identify those attribute identifiers in the AS of the RDS which appear in our query. In that case, a_1, a_2, a_3, and a_5 (KEGG) and b_1, b_2, b_3, and b_4 (BRENDA) must be retrieved from the RDS. Initially, each RDS addresses the query on its own. That is, the retrieval dependencies are being exploited to answer the query locally. The RDS establishes an empty attribute set (here $R_{\mathrm{K}}(a_1, a_2, a_3, a_5)$ and $R_{\mathrm{B}}(b_1, b_2, b_3, b_4)$) which gathers the query result. The KEGG data source facilitates the opportunity to retrieve the set of all stored values of a_1 (rule $H_{\mathrm{KEGG},1}$) and, initially, fills R_{K} as follows:

$$R_{\mathrm{K}} = \{(1.1.1.1, \_, \_, \_), (1.1.1.2, \_, \_, \_), (1.1.1.3, \_, \_, \_)\}$$

The "_" mark identifies a null value. The rule $H_{\mathrm{KEGG},2}$ relates a_1 to a_2 and a_3. That is, for each attribute value a_1 in R_{K} we retrieve the appropriate a_2 and a_3 values:

$$
\begin{aligned}
R_{\mathsf{K}} = \{&(1.1.1.1, \mathsf{ADH}, \mathsf{prim.\ alc.{+}NAD{+}{=}ald.{+}NADH{+}H{+}}, \_),\\
&(1.1.1.1, \mathsf{ADH}, \mathsf{sec.\ alc.{+}NAD{+}{=}ketone{+}NADH}, \_),\\
&\ldots\\
&(1.1.1.2, \mathsf{NADP}, \mathsf{prim.\ alc.{+}NADP{+}{=}ald.{+}NADPH}, \_),\\
&\ldots\\
&(1.1.1.3, \mathsf{HSDH}, \mathsf{L\text{-}homoser.{+}NAD{+}{=}L\text{-}asp.\ 4\text{-}semiald.{+}NADH}, \_),\\
&(1.1.1.3, \mathsf{HSD}, \mathsf{L\text{-}homoser.{+}NADP{+}{=}L\text{-}asp.\ 4\text{-}semiald.{+}NADH}, \_)\}
\end{aligned}
$$

Obviously, each attribute value of a_1 (EC number) yields several distinct values of a_2 (enzyme name) and a_3 (reaction). For example, the EC number "1.1.1.1" represents an enzyme abbreviated as "ADH" or "NAD" (actually, there are even more synonyms) which is involved (as substrate or product) in three chemical reactions. Thus, we obtain six combinations of enzyme names and reaction involvement for this very EC number which similarly applies to the other enzymes.

Unfortunately, there is no retrieval dependency that yields the remaining attribute a_5 (pathway name) by any of the hitherto determined attribute values (a_1, a_2, or a_3). Instead, it appears in rule $H_{\mathrm{KEGG},5}$ which requires a_4. We follow a top-down approach to construct the retrieval dependency graph.

First, we identify all rules that yield a_5. In this simple scenario, there is only one rule where a_5 appears on the right side ($H_{\mathrm{KEGG},5}$). In case the "input attribute" of any of these rules had been retrieved already (a_1, a_2, a_3) or there was no input attribute (like in $H_{\mathrm{KEGG},1}$ and $H_{\mathrm{KEGG},4}$) and no attribute was retrieved as yet, we could return this rule as one-element dependency chain and stop. In our case, however, a_4 appears as input attribute on the left side of $H_{\mathrm{KEGG},5}$ and has not yet been retrieved. We assign a node for $H_{\mathrm{KEGG},5}$ and recursively identify all rules where a_4 appears on the right side. In our case this is $H_{\mathrm{KEGG},4}$ and $H_{\mathrm{KEGG},3}$. Suppose, $H_{\mathrm{KEGG},4}$ was identified first. In that case, a list of all attribute values of a_4 could be retrieved by the system. As we already retrieved the attribute values of a_1, a_2, and a_3, this rule is of no use since it does not properly relate a_1, a_2, a_3, and a_4. Thus, we discard $H_{\mathrm{KEGG},4}$ and examine $H_{\mathrm{KEGG},3}$, where a_1 appears on the left side. Obviously, a_1 is among the hitherto retrieved attributes. We, thus, assign a node for $H_{\mathrm{KEGG},3}$, draw a directed edge between $H_{\mathrm{KEGG},3}$ and $H_{\mathrm{KEGG},5}$, return the emerging dependency chain $\{H_{\mathrm{KEGG},3}, H_{\mathrm{KEGG},5}\}$, and stop.

Although we found this simple scenario to be fairly typical for many data sources, more complex situations may arise. For example, an attribute might not be retrievable at all. In that case, the algorithm would run until it would not find a rule which has a retrievable attribute on the left side, report an error, and stop. A more difficult situation arises if the recursive traversal of dependency rules formed cycles. In our example, suppose, we arrive at $H_{\mathrm{KEGG},3}$ where the input attribute a_1 had not yet been retrieved before. We might, thus, consult $H_{\mathrm{KEGG},5}$ again, where a_1 appears on the right side and run into an endless loop.

To tackle this problem, we exclude any node from being visited twice. That is, any rule which has been used before (i.e. appears as a node in the dependency graph) can not be used again. The third problem occurs for dependency rules whose input attribute already appears on the right side of some other node. We may take advantage of this situation by directly establishing an edge between both nodes. However, to not run into cycles, and, thus, contradict our second postulation, we restrict this simplification to disjoint branches of the dependency graph.

Coming back to our example, we have identified a dependency chain which (i) retrieves a_4 by a_1 and then (ii) retrieves a_5 by a_4. Here a_4 serves as some temporary helper attribute (appears in brackets in R_{K}):

$$
\begin{aligned}
R_{\mathsf{K}} = \{\ldots \\
&(1.1.1.3, \mathsf{HSDH}, \mathsf{L\text{-}homoser.{+}NAD{+}{=}L\text{-}asp.\ 4\text{-}semiald.{+}NADH}, (\mathsf{MAP00260})) \\
&(1.1.1.3, \mathsf{HSD}, \mathsf{L\text{-}homoser.{+}NAD{+}{=}L\text{-}asp.\ 4\text{-}semiald.{+}NADH}, (\mathsf{MAP00260})) \\
&(1.1.1.3, \mathsf{HSDH}, \mathsf{L\text{-}homoser.{+}NAD{+}{=}L\text{-}asp.\ 4\text{-}semiald.{+}NADH}, (\mathsf{MAP00300})) \\
&(1.1.1.3, \mathsf{HSD}, \mathsf{L\text{-}homoser.{+}NAD{+}{=}L\text{-}asp.\ 4\text{-}semiald.{+}NADH}, (\mathsf{MAP00300}))\}
\end{aligned}
$$

Due to space limitations, we have restricted ourselves to the enzyme number (a_1) "1.1.1.3" which appears in two pathways identified by the maps (a_4) "MAP00260" and "MAP00300". In the next step, this helper attribute gets replaced by a_5 through rule $H_{\mathrm{KEGG},5}$:

$$
\begin{aligned}
R_{\mathsf{K}} = \{\ldots \\
&(1.1.1.3, \mathsf{HSDH}, \mathsf{L\text{-}homoser.{+}NAD{+}{=}L\text{-}asp.\ 4\text{-}semiald.{+}NADH}, \mathsf{glyc.,\ ser.,\ threon.\ metab.}), \\
&(1.1.1.3, \mathsf{HSD}, \mathsf{L\text{-}homoser.{+}NAD{+}{=}L\text{-}asp.\ 4\text{-}semiald.{+}NADH}, \mathsf{glyc.,\ ser.,\ threon.\ metab.}), \\
&(1.1.1.3, \mathsf{HSDH}, \mathsf{L\text{-}homoser.{+}NAD{+}{=}L\text{-}asp.\ 4\text{-}semiald.{+}NADH}, \mathsf{lys.\ biosynthesis}), \\
&(1.1.1.3, \mathsf{HSD}, \mathsf{L\text{-}homoser.{+}NAD{+}{=}L\text{-}asp.\ 4\text{-}semiald.{+}NADH}, \mathsf{lys.\ biosynthesis})\}
\end{aligned}
$$

Finally, $R_{\mathrm{B}}(b_1, b_2, b_3, b_4)$ has to be filled by the attribute values of BRENDA. Unfortunately, BRENDA does not provide any mechanism to retrieve a list of all attribute values. We must, thus, provide an initial attribute list externally. Though the attribute identifiers between the KEGG and BRENDA adapter schema and the IUS are disjoint, our *same*-relations serve as a solution to this problem. In our case we should find a *same*-attribute to b_1 which appears only in the dependency rule of BRENDA, $H_{\mathrm{BRENDA},1}$. In fact, a_1 of KEGG is a *same*-attribute of b_1. We may either directly facilitate $H_{\mathrm{KEGG},1}$ to retrieve the a_1 values or, alternatively, use the distinct occurrences of a_1 from R_{K}, i.e.:

$$R_{\mathrm{B}} = \{(1.1.1.1, \_, \_, \_), (1.1.1.2, \_, \_, \_), (1.1.1.3, \_, \_, \_)\}$$

Next, we employ $H_{\mathrm{BRENDA},1}$ to retrieve and combine the remaining attributes b_2 (name), b_3 (reaction), and b_4 (organism):

$R_{\mathrm{B}} = \{\ldots$
(1.1.1.3, HDH, L-homos.+NAD+=L-asp. 4-semiald.+NADH, aquif. aeolicus),
(1.1.1.3, HSD, L-homos.+NAD+=L-asp. 4-semiald.+NADH, aquif. aeolicus),
(1.1.1.3, HDH, L-homos.+NAD+=L-asp. 4-semiald.+NADH, helicob. pylori),
(1.1.1.3, HSD, L-homos.+NAD+=L-asp. 4-semiald.+NADH, helicob. pylori)$\}$

Finally, the actual integration task must be conducted. That is, a join operation can be performed to obtain the desired enzyme information. A natural join would restrict the result to those enzymes which appear in both data sources, and, thus provide a complete information coverage for the integrated data. Alternatively, an outer join preserves all information from any data source but does eventually not provide all attribute values for each enzyme which would be expressed as null values. Based on the semantics of the application scenario, the user must choose between both alternatives. The second choice is on the join attributes. Obviously, any combination of *same*-attributes that appear in R_{K} and R_{B} may be employed. In our case a_1 (b_1), a_2 (b_2), and a_3 (b_3) fulfill this requirement. We, therefore, join on the attribute combinations (a_1, a_2, a_3) and (b_1, b_2, b_3). We (naturally) join the R_{K} and R_{B} and obtain:

Enzyme $= \{\ldots$
(1.1.1.3, HSD, L-homos.:NAD+=L-asp. 4-semiald.+NADH,
glyc., ser., threon. metab., aquif. aeolicus),
(1.1.1.3, HSD, L-homos.:NAD+=L-asp. 4-semiald.+NADH,
glyc., ser., threon. metab., helicob. pylori)$\}$

For the enzyme "1.1.1.3" (*homoserine dehydrogenase*) two tuples exist in the *Enzyme* attribute set of the integrated schema. That is, R_{K} and R_{B} agree on a_1, a_2, a_3 (b_1, b_2, b_3) for these tuples, only. An alternatively applied outer join integration operation would preserve all information gathered in R_{K} and R_{B} but cause null-values in the *pathway* or *species* attributes for many tuples.

5 Sample Applications

An application demonstrating the system's capabilities should be as simple as possible in order to minimize barriers of use. This motivated the application *DBOra* that will enable the user to browse through the data. Currently, our integrated data stock comprises about 10 GByte (35 million tuples) of data stored in approximately 80 relations cross-linked through foreign keys. A second application, *MailSSA*, enables the user to select a certain starting and end table from the integrated data schema and to collect the interconnected information in between by querying the start table for a certain string or sequence and relating those data with data connected from the end table. Both applications are accessible via www-bm.ipk-gatersleben.de.

In the following we restrict ourselves to describe the functions of DBOra since it particularly highlights to afore mentioned capabilities of the integration. Deploying DBOra to navigate through the integrated data starts with either a text search over all integrated data entries or with a *BlastP* search for amino acid (AA) sequences or *BlastX* for nucleotide acid (NA) sequences, respectively. The result set will return all entries that include the initial search term(s) or provide a Blast hit of the AA or NA sequence. Hence the user can select any of those tables to display the entries more detailed. Should he find interesting data he will be able to search for data entries in other categories (tables) that might enhance and complement the already discovered data, provided that data exists of course. Furthermore, the user can restrict his search to data entries that are located in neighboring or directly connected tables with respect to the database schema of the integration system. This feature might be especially rewarding to users who formerly invested time to study the database schema and to identify relevant and irrelevant relations between the included database tables. Whereas for users investing that time, it might be more appropriate to use the option to display all information that can be reached from the current data entry by following the links between all relevant tables of the schema. The corresponding result set will be directly displayed as detailed information. With the option to display all interconnected data, all tables, neighboring and remote via corresponding neighbors, the user can search for entries in the integrated database.

6 Outlook

Finally, we outline some ideas on future progresses to improve and complement the *BioDataServer* integration system. We can basically distinguish our ongoing work into two fields which are centered around (i) the development of applications that interact with the integrated data and (ii) improvements of the adapter components.

As for the application development, we are currently investigating possibilities on how to utilize our integrated data in connection with identifying coding regions in ESTs by information retrieval mechanisms. Currently, a research project at IPK gathers information from various resources like protein domain, transcription factor, signal peptide, and subcellular localization databases to perform a statistical analysis for automatic EST annotation.

Though we focused on the overall architectural design and query processing, semi-automatic adapter implementation support plays an important role in minimizing the cost to introduce a new data source. Research in bioinformatics has come up with many existing tools to query and extract data from various popular data sources. To avoid re-implementations of these tools we currently develop a re-use concept. Though this approach permits a re-use of certain implementation components, most of the adapter programming work is still a tedious, error-prone task. We therefore, investigate learning mechanisms to detect the structure of a data source semi-automatically.

References

1. F. Achard, G. Vaysseix, and E. Barillot. XML, bioinformatics and data integration. *Bioinformatics*, 17(2):115–125, 2001.
2. B. Boeckmann, A. Bairoch, R. Apweiler, M.-C. Blatter, A. Estreicher, E. Gasteiger, M. J. Martin, K. Michoud, C. O'Donovan, I. Phan, S. Pilbout, and M. Schneider. The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. *Nucleic Acids Research*, 31(1):365–370, 2003.
3. T. Etzold, A. Ulyanow, and P. Argos. SRS: Information Retrieval System for Molecular Biology Data Banks. *Methods in Enzymology*, 266:114–128, 1996.
4. A. Freier, R. Hofestädt, M. Lange, U. Scholz, and A. Stephanik. BioDataServer: A SQL-based service for the online integration of life science data. *In Silico Biology*, 2(0005), 2002. *Online Journal*: `http://www.bioinfo.de/isb/2002/02/0005/`.
5. L. M. Haas, P. M. Schwarz, P. Kodali, E. Kotlar, J. E. Rice, and W. C. Swope. DiscoveryLink: A system for integrated access to life sciences data sources. *IBM Systems Journal*, 40(2):489–511, 2001.
6. A. Hamosh, A. F. Scott, J. Amberger, C. Bocchini, D. Valle, and V. A. McKusick. Online Mendelian Inheritance in Man (OMIM), a knowledgebase of human genes and genetic disorders. *Nucleic Acids Research*, 30(1):52–55, 2002.
7. M. Höding. *Methoden und Werkzeuge zur systematischen Integration von Dateien in Föderierte Datenbanksysteme*. Shaker Verlag, Aachen, 2000.
8. W. H. Inmon. *Building the Data Warehouse*. John Wiley & Sons, Inc., 2 edition, 1996.
9. M. Kanehisa, S. Goto, S. Kawashima, and A. Nakaya. The KEGG database at GenomeNet. *Nucleic Acids Research*, 30(1):42–46, 2002. http://www.genome.ad.jp/kegg/.
10. P. D. Karp. A Strategy for Database Interoperation. *Journal of Computational Biology*, 2(4):573–586, 1995.
11. Steven Prestwich and Stéphane Bressan. A SAT Approach to Query Optimization in Mediator Systems. In *Proceedings of the Fifth International Symposium on the Theory and Applications of Satisfiability Testing (SAT 2002), Cincinnati, Ohio, USA*, pages 252–259, 2002.
12. I. Schomburg, A. Chang, and D. Schomburg. BRENDA, enzyme data and metabolic information. *Nucleic Acids Research*, 30(1):47–49, 2002.
13. A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.
14. A. Siepel, A. Farmer, A. Tolopko, M. Zhuang, P. Mendes, W. Beavis, and B. Sobral. ISYS: a decentralized, component-based approach to the integration of heterogeneous bioinformatics resources. *Bioinformatics*, 17(1):83–94, 2001.
15. R. Stevens, P. Baker, S. Bechhofer, G. Ng, A. Jacoby, N. W. Paton, C. A. Goble, and A. Brass. TAMBIS: Transparent Access to Multible Bioinformatics Information Sources. *Bioinformatics*, 16(4):184–185, 2000.
16. T. A. Tatusova, I. Karsch-Mizrachi, and J. A. Ostell. Complete genomes in WWW Entrez: data representation and analysis. *Bioinformatics*, 15(7/8):536–543, 1999.
17. G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3):38–49, March 1992.