appeared in: First IFCIS International Conference on Cooperative Information Systems, Brussels, Belgium, June 19–21, 1996

A Methodology for Building a Data Warehouse in a Scientific Environment

Karl Aberer, Klemens Hemm
GMD–IPSI
Dolivostr. 15, 64293 Darmstadt, Germany
{aberer, hemm}@darmstadt.gmd.de

Abstract

Rational drug design is an example where integrated access to heterogeneous scientific data is urgently needed, as it becomes rapidly available due to new experimental and computational techniques. This is currently problematic as data is scattered over heterogeneous, mostly file-based legacy databases, the data is erroneous, incomplete, and inconsistent in representation and content. We have developed a methodology, including metadata specification, data transformation and software architecture, that supports data analysis and preparation for building a data warehouse using object-oriented database technology. With this methodology the system ReLiBase has been realized, that allows querying and visualization of the drug-design related data from heterogeneous resources.

1 Introduction

Developing a new drug is both an expensive and time consuming enterprise. Traditional approaches like modifying a well known drug or performing random screening over huge sets of organic molecules lack controlled result orientation and substantial innovative perspectives. 'Rational drug design' tries to obtain a functional understanding of the processes in an organism, and to propose new lead substances for drugs. The starting point for the study of biochemical processes are databases of experimental data. Analysis and interpretation of this 'raw data', involving complex computational tools, guide the drug design cycle.

In cooperation with the pharmaceutical companies Merck, Darmstadt, and BASF, Ludwigshafen, and the European Molecular Biology Laboratory EMBL, Heidelberg, we have developed ReLiBase, a tool to provide drug designers with an integrated platform supporting powerful access mechanisms to qualitatively enriched data relevant for drug design. (The work is performed within the national joint project RELIWE "Computation and Prediction of Receptor-Ligand Interaction" and supported by the German "Bundesministerium für Bildung und Wissenschaft", BMBF, grant number 01IB302E). Two central questions had to been answered when designing the system. What is the appropriate platform and what is the methodology to migrate data from the various source databases to the target platform. With regard to the first question it is in the meanwhile well accepted wisdom in biomolecular applications, that object-oriented systems are the appropriate choice due to the ability to flexibly represent the various domain-specific data types and algorithms. An indication for this is the use of object-oriented technology in various projects related to biomolecular databases[10][13][18]. Therefore we are using the object–oriented database management system VODAK [24] as data warehousing platform. So a main focus has been to develop an integration methodology.

Database integration is a well studied problem in computer science [25]. Approaches range from classical federated database architectures, like in the IRO–DB [12] approach for integrating relational and object-oriented databases, to data warehousing in commercial settings, like reported in [15]. In [7] it has been noted that the issues occurring in the biomolecular domain differ with regard to the complexity of queries and data modelling. Thus approaches with the background of commercial applications have only limited applicability. The authors of [7] propose to access biomolecular data based on an object-oriented data model without physically integrating the data. Queries are expressed in a comprehension language, are decomposed, optimized and delegated to various external databases and can perform the required data transformations. Although this approach is striking for its flexibility it is less applicable to our problems for performance considerations.

For obtaining adequate performance it was quite clear from the beginning that for data analysis purposes at least a core database has to be integrated physically, in other words a data warehouse [15] was required. The observation that substantial biochemical and biomolecular knowledge is required to appropriately prepare the data for the warehouse was identified as a main problem. This includes selection of data, correction of errors, addition of missing data or consolidation of data from different databases. The knowledge for data preparation is primarily provided at the side of the industrial partners. On the other hand integrating the data from various file-based databases to an object-oriented database requires non-trivial data management techniques. We had to resolve the question how expert knowledge from both fields could be smoothly combined.

Existing approaches on data warehousing in molecular biology are lacking a methodology in the support for data preparation and analysis. An example for integrating and enriching file-based databases for covering a particular specialized aspect, namely protein sequences, is SWISSPROT [5], itself a file-based database. Consistency is there maintained by an editorial team. An approach for building a warehouse for genome data in an automatized way is IGD [23]. Heterogeneous data is first represented in proprietary object-oriented structural model (ACE [10]). Integration is performed by matching semantically equivalent entities and resolving conflicts by juxtaposition, on schema as well as on instance level. Thus integration and correction based on data analysis is not performed. Finally the object-oriented schemas are automatically translated into relational schemas for building the warehouse on a relational system (Sybase). Other warehouses store only the information that is required to link information from different databases, like SRS [11].

The contribution of this paper is a methodology for building a data warehouse that has evolved from the close cooperation with industrial scientific users (mostly biochemists and molecular biologists, we will refer to them in the following as ”the users”). It has proven to be a practicable approach in building the data warehouse for their needs. The key components of this methodology are

1. A well defined interface between the users and the computer scientist in form of a *metadatabase*, that is specified by the users. This metadatabase allows the specification of the data that is to be included into the data warehouse, of how data from different databases is to be interconnected, and of additional data that is not contained in one of the source databases. The operational execution of this specification is the task of the data management system.

2. A *basic set of data transformations* that are on the one hand sufficiently expressive to transform the heterogeneous source databases to a target database, but are on the other hand sufficiently simple in semantics such that a biological user can understand their effects without having knowledge of technical details of the target database or the transformation process.

3. The *software design* to execute the data transformations from file-based source databases to the object-oriented target database as specified in the metadatabase in an efficient and automatized manner.

As an application of these concepts *ReLiBase*, a data warehouse for rational drug design, has been built based on this methodology. The concepts have proven to be easily applicable by the users and the separation of the aspects of data analysis in a preparatory phase and the actual physical building of the data warehouse has been successfully performed. Currently the data warehouse is under evaluation by the users.

In this paper we first analyze in Section 2 requirements with regard to data integration that were derived during the problem analysis. The key contributions of the paper are found in Section 3. There we formally analyze the data transformations, that are required, introduce the concept of using a metadatabase for data warehousing, and discuss the software architecture we have chosen. In Section 4 we discuss the implementation of the system. In Section 5 some concluding remarks are given.

2 Requirements for data warehousing

We mention some of the important data resources used in our work. Detailed structural data on proteins and protein–ligand complexes is available in PDB [6]. This database contains currently about 3000 protein structures. In the year 2000 it is expected to contain about 30.000 structures. Structures are used for detailed analysis of protein–ligand interaction. More than 100.000 protein sequences are available in public databases, like SWISSPROT [5]. Mutation data allows to identify functionally critical regions in proteins. This information is found in SWISSPROT and in specialized databases like PMD [20]. The different databases and corresponding tools interoperate only little if at all. The hope is to obtain new insights in the interaction of biological molecules with drugs by enabling the analysis of data from different resources and thus combining different paradigms (e.g. key-lock principle, sequence similarity, mutation).

The existence of these databases in public form is desirable from the viewpoint of availability. However, since the data providers are autonomous a high degree of heterogeneity is encountered. Most data is stored in file-based databases. One often encounters semantically complex data in poorly structured representations. Database schemas are often missing or implicit. Data is often incomplete and erroneous. Data may only be available as the result of analysis programs or even only accessible in the literature. Thus to provide quality data for the scientific work an extensive analysis and enrichment process is required. To illustrate the difficulty in integrating the data we sketch a few steps that need to be realized within our project for preparing the data warehouse (for a more extensive description see [14]).

Starting from the set of existing PDB files those are identified that contain a protein. If this is the case it is tested whether a ligand exists, that is not another protein or a nucleic acid. Only in this case a protein object, a complex object and a ligand object are created and the complex is connected with the corresponding protein and ligand. The three dimensional complex models in the PDB file are identified and are added to the data warehouse. Now the topology of the ligand is constructed as it is not available in PDB. This step requires algorithmic tools, access to ligand databases and manual inspection and includes error correction and elimination of duplicate structures. When the corresponding structure is identified, a file in MOL2 format is added to the source database and the model needs to be connected to the corresponding ligand. For the protein the SWISSPROT database is searched for a sequence entry. This is done either by using keyword search or by using a sequence similarity database. If it is found the SWISSPROT entry is created and connected to the corresponding protein. Possibly there are discrepancies in the numbering schemes of the SWISSPROT sequence and the sequence stored in PDB. In this case corresponding corrections are generated to align the residue numbering of swissprot with the numbering used in the PDB model. From SWISSPROT references to other sequence databases, like PIR or EMBL, can be derived. Mutation data can be derived either from the SWISSPROT database or the PMD database. The process is based on different stand-alone programs and parsers, and the different steps evolve continuously.

The following requirements have been derived with regard to building the data warehouse. It is necessary

(5) to access heterogeneous file-based databases e.g. by parsing techniques.

(6) to perform complex data restructuring in order to transform the data from the different heterogeneous data formats into the integrated warehouse.

(7) to support correction of errors, data adjustments and data completion by using complex data analysis techniques.

(8) to be able to track any data back to its original resources for verification or visualization.

(9) to allow the user to flexibly adapt the scope, structure and contents of the data warehouse to his needs without requiring support from the data management expert.

(10) to relieve the user as far as possible from details on a technical level of integration.

(11) to minimize manual intervention as far as possible during the data integration process.

(12) to consider updates in external formats and schema evolution.

Some of the requirements are contradictory such that a feasible compromise needs to be found. For example making the user able to adapt the integration process (5) is in conflict with requiring minimal technical understanding from his side (6). Or the desire to fully automatize the process (7) opposes the requirement that manual intervention might be required (3). Some aspects simplify the integration methodology, in particular the fact that we do not have to deal with continuous updates of the source databases, but rather with different release policies.

Of course we were not able to cover all of the above requirements completely or equally well. However we carefully considered these requirements in the system design such that we do not introduce artificial barriers for future steps towards the ”ideal” system.

3 Building the data warehouse

We have developed an application-oriented approach for a data integration methodology that constitutes the main focus of this paper. Although the approach is not generic in implementation it appears to us, that it covers some basic aspects, which are of interest for the development of a general data warehouse integration methodology.

3.1 Preliminary observations

It is in the meanwhile a well accepted fact that some kind of object-oriented system is required [1][16] for efficient handling of the complex, application-specific data structures that occur in molecular biology. As we are dealing with a huge amount of data the system must support secondary storage management and multi-user access, thus object-oriented database management system are the favorable choice.

From the viewpoint of schema integration source and target schemas are predetermined in our application scenario. For the source schemas this is obvious as we are dealing either with public or proprietary databases that have full autonomy.

For the design of the object-oriented target schema an informal approach was chosen by exchanging on the one hand implicit knowledge and preferences of the users and on the other hand the knowledge of the computer scientists on the feasibility of certain data representations within the chosen object-oriented DBMS. In this way the desired target schema evolved quite rapidly. It reflects the particular view and preferences of drug designers with regard to data and tools they want to employ in their research.

What remains to be solved is the data transformation and integration problem. Data from a number of source databases has to be transformed into one target database. Conceptually we can interpret the source databases as instances of particular object-oriented schemas. Most of the databases are organized in files, each file corresponding to one (or a few) objects, e.g. proteins, sequences, ligands. The data in files are organized in record structures. Specific records may be optional. Values can be simply structured data or can be texts, sequences, or again record structures (e.g. the coordinates of an atom). For a formal description of the data transformation process it is however useful (and feasible) to map these structures into a structural object-oriented data model and make implicit data structures in this way explicit. In principle, one could perform this step by defining import schemas, like in a federated DBMS architecture [25], and then perform a semantic integration of the data on top of these import schemas. For performance considerations we abandoned this possibility. Instead we provide corresponding access mechanisms directly on the files which are typically based on parsers.

In the transformation between source and target databases complex restructuring of the data may occur, including splitting and merging of objects, extracting keywords from text segments or restructuring of sequences. Those structural relationships between source databases and the target database are sometimes relatively clear. For example, it is straightforward to transform a given tuple representing the coordinates of an atom to a corresponding atom object in the target database, although the structural representation might be different. Other aspects of the data transformation cannot be easily described as data restructuring operations. For example a PDB file may not contain data on proteins at all, but rather on nucleic acids. Or it may contain in addition to protein data also protein ligand data. In such a case the ligand may be another protein (a so-called peptidic ligand), a nucleic acid, or a small organic molecule. Such distinctions can only be made by using heuristic algorithms that reflect substantial biochemical knowledge and they are required order to correctly identify those objects to which subsequent transformations can be applied correctly. Thus, the problem of integrating data is not only one of data model transformation and data restructuring, but requires semantic analysis of data. Therefore different tasks in the integration process allow for different degrees of automation.

3.2 Language for describing data transformations

In order to describe the data transformation process formally we will adopt the language WOL [19] for expressing data transformations.

The language is based on a simple structural object-oriented data model with object-identities, classes and complex data-structures, including records and sets. We introduce the main concepts by means of a few examples. Let a schema be given for representing data from a PDB file with one class *PDB_file*. The class has associated a type that matches the file format, for example

$\text{type}_{\text{PDB\_file}}$ =
    (name: STRING,
    atoms: { [column_1: STRING, column_2: STRING,
                 column_3: STRING, column_4: STRING] })

An instance of this class can be considered as an abstract representation of a concrete PDB file. For example the instance

```
PDB_file("1atr", { ["N", 12.1, 1.7, 14.5], ["C", 0.7,  3.5,  23.5],... })
```

matches the contents of a file with name *1atr* and the following content:

```
ATOM            N          12.1 1.7 14.5
ATOM            C          0.7  3.5  23.5 ....
```

The four record entries correspond of course to the atom type and the 3D atom coordinates. We also consider a second schema for the representation of a protein model in an object-oriented database.

$\text{type}_{\text{ProteinModel}}$ =
    (proteinname: STRING, atoms: {Atom})
$\text{type}_{\text{Atom}}$ =
    (atomtype: STRING,
    coordinates: [x: REAL, y: REAL, z: REAL])

Now we give a simple example of a constraint that could hold between these two schemas

$P \in \text{ProteinModel}, P.\text{proteinname} = X.\text{name} \Longleftarrow X \in \text{PDB\_file}$

The capital letters *X* and *P* denote variables and the dot notation expresses attribute access to records in the usual way. Class names denote the set of instances of the class. The rule is interpreted as follows. For some instantiation of the variables that makes the condition on the right hand side true the left hand side has also to be true. This rule describes an inter-database constraint containing a condition on objects from the target database on the left hand side and a condition on objects of the source and target database on the right hand side. Such a rule is called a transformation rule.

Assuming that the source database(s) is populated, and that the right hand side matches the given pattern (in our example $X \in$ *PDB_file*), and further assuming that no object exists that satisfies the condition on the left hand side, an action has to be performed in order to satisfy this constraint. For example, this action can be the creation of an appropriate object in the target database (in our example the creation of a *ProteinModel* object) or the update of an attribute of such an object (in our example setting the property of the created object to *X.name*). Thus, in order to satisfy this constraint a data transformation is performed.

The same kind of rules can be used to define intradatabase constraints, in the case where only classes of one database are involved. Then, for example, a DBMS would be required to maintain these constraints. A simple example is

A.atomtype ∈ {"C", "N", "O", ... } <== A ∈ Atom

In this paper we do not assume, that a generic implementation of the WOL language exists, rather we will implement different rules by different specialized programs. However, it is valuable to express the transformations we use in a formal way in order to analyze the different steps of the data warehouse building process.

3.3 The use of metadata for data integration

We continue the example of the previous section. In principle, the database transformation for protein models can be specified by a single rule.

```
P ∈ ProteinModel, P.proteinname = X.name, A in P.atoms,
     A.name = Y.column1, A.coordinates.x = Y.column2,
     A.coordinates.y = Y.column3, A.coordinates.z = Y.column4
     <==  X ∈ PDB_file, Y ∈ X.atoms, is_protein(X)
```

However, not all PDB files contain protein models. This requires the evaluation of a predicate *is_protein*, which hides an analysis of the source data. For example to decide whether the PDB file contains a protein or a nucleic acid, a statistical analysis of the occurring atom types is required. This process cannot be described within the data transformation language (and it is not obvious how any general purpose formalism can do this). The evaluation of certain predicates even may not allow for automated processing. For example, when transforming small molecule structures from PDB files into topological representations with binding patterns, any algorithm must be manually supervised and requires eventually corrective actions. Although we are able to describe now the transformation from the source to the target database on an abstract level, it does not adequately describe what happens "behind the scenes".

Thus we proceed in two steps. First we identify those components of the data transformations that can be fully described as data restructuring. For these components we develop the specification of the data transformation in form of transformation rules and provide functions that can perform the transformations. For example, this is feasible, when a PDB file is identified to contain exactly the model of one protein, then this file can easily be transformed to an equivalent representation in the data warehouse. This kind of transformations then provides a set of *atomic data transformations* that can serve as components in the whole transformation process.

For those components of the data transformations that cannot be treated in this way, we define a *metaschema* that contains additional information. This information is derived by data analysis, like the determination of the value of the predicate *is_protein*. The corresponding metadatabase is prepared by users and treated in the data transformation process as additional source database. Supplemented with this metadatabase the whole data transformation process can be performed fully automatically (see req (7)).

The metadatabase defines a "parameter space", in which the user can control the integration process and define the contents of the data warehouse (see req (5)). From another perspective the metadatabase establishes a well-defined and sufficiently restricted interface between the computer scientists and users. It factors out those aspects of data integration, that are not well definable, after having agreed on stable knowledge which is encoded in atomic data transformations. This corresponds to a division of work, where computer scientists take care of data restructuring and consistency management and users take care of the application-specific aspects that occur in data integration. The computer scientist and the user thus exchange only information on aspects that are really variable in the data transformation scenario. Another benefit of using metadata is that it allows the user to define different data warehouses for specialized purposes. A similar conception using metadata is considered e.g. in [15].

We illustrate the concept now by proceeding with the example. We assume the following class is given in the metaschema

$\text{type}_{\text{ProteinModel\_meta}}$ = (proteinname: STRING)

Then the user generates as a result of data analysis the following object

ProteinModel_meta("1atr")

where *1atr* is the name of a protein and its corresponding PDB file. The data transformation process is then described by the following rule

```
P ∈ ProteinModel, P.proteinname = Z.proteinname, A in P.atoms,
     A.name = Y.column1, A.coordinates.x = Y.column2,
     A.coordinates.y = Y.column3, A.coordinates.z = Y.column4
     <==  X ∈ PDB_file, Z ∈ ProteinModel_meta,
          Z.proteinname = X.name, Y ∈ X.atoms
```

In this rule predicates with "hidden meaning" do not occur anymore. Still, the one step transformation rule does not appropriately reflect the work division that was discussed previously. To achieve this we decompose the rule into a rule corresponding to an atomic data transformation, independent of metadata, and a rule corresponding to the transformation of metadata. The latter can be considered as *composite data transformation*, since it triggers subsequent atomic transformations by changing the contents of the target database. The atomic data transformation rule is

```
A ∈ P.atoms, A.name = Y.column1, A.coordinates.x = Y.column2,
     A.coordinates.y = Y.column3, A.coordinates.z = Y.column4
     <==  P ∈ ProteinModel, X ∈ PDB_file,
          P.proteinname = X.name, Y ∈ X.atoms
```

This rule expresses, that when a protein model object exists in the target database, then the corresponding attributes, in this case the information on atoms, can be automatically retrieved from the corresponding source PDB file (in practice by using an appropriate file parser). It is important to observe that this rule is independent of the metaschema.

The transformation of metadata on the other hand is described by the following rule

$P \in$ ProteinModel, P.proteinname = Z.proteinname
  <== $Z \in$ ProteinModel_meta

This rule states whenever there exists an entry in the metadatabase stating the existence of a protein model then a corresponding entry in the target database for the protein model is required. In this way we have organized the transformations in a way such that the atomic transformation are specified independently of the transformations of metadata. As both kinds of transformations are described in a simple data restructuring language it is obvious that they can be systematically implemented.

To conclude this section we illustrate the usage of rules for the description of target database constraints and metadatabase constraints. Both kinds of constraints play actually a role in our application. A typical case for such constraints are primary keys which play a central role in the identification of objects during integration. For keys the language WOL provides a special function $Mk_C(t)$ which represents the object identity of the object *o* in class *C* with key *t*, where *t* is a term of appropriate type. Then we can state the following constraints

$P = Mk_{ProteinModel}$(P.proteinname) <== $P \in$ ProteinModel
$Z = Mk_{ProteinModel\_meta}$(Z.proteinname)
  <== $Z \in$ ProteinModel_meta

Actually these and other constraints are checked in the course of the data transformation process.

To summarize a metadatabase is prepared by the user providing all the information to transform and interconnect the information from the different databases and to correct the data in the desired way. This specification fully describes the data warehouse in a declarative way, from which an automatic process can physically build it.

For the sake of clarity, the examples used in the presentation of this section are to a certain degree a simplification of the situation we encounter in practice. We will give in the following section examples of more complex transformations.

3.4 Types of transformations

To keep the design of transformation rules maintainable and the semantics of the metadatabase transparent for the user, who has to generate this metadata, we have restricted the types of possible transformation rules applicable to the metadatabase. Experience has shown that in our application scenario the four following rule types are sufficient:

(1) *Creation of an object in the target database:* It has the general format

$O \in$ C_target, *(clauses for initialization of attributes)*
  <== $X \in$ C_meta

For example for identifying the proteins for the data warehouse we introduce a new type for the metadatabase

$type_{Protein\_meta}$ = (proteinname: STRING)

where the corresponding transformation rule is

$P \in$ Protein, X.proteinname = P.proteinname
  <== $X \in$ Protein_meta

(2) *Change of an attribute of an existing object in the target database:* These rules are used to set attributes that need to be changed only occasionally and are thus not set at object creation time. A typical application of this rule is the correction of errors or complementation of missing data in the source database (see req (3)). These rules have the general format

(clauses for initialization of attributes)
  <== $X \in$ C_meta, $O \in$ C_target, X.key = O.key

where *key* is a key attribute of the corresponding class. A concrete example of such a rule is

X.organism $\in$ Protein.organism
  <== $X \in$ Protein_organism_meta, $O \in$ Protein,
    X.proteinname = O.proteinname

where the type in the metadatabase for describing such corrections is

$type_{Protein\_organism\_meta}$ =
  (proteinname: STRING, organism: STRING)

and a sample entry in the metadatabase is

Protein_organism_meta("1atr","mouse")

It is frequently the case that annotation information, like in the example, is missing in the PDB database and needs to be added by the user.

(3) *Creation of existence-dependent objects in the target database:* In many cases we encounter part-of relationships in the target schema. For these it is advantageous to provide an appropriate transformation rule type.

$P \in$ C_target_part, P.whole = O, $P \in$ O.parts,
  (clauses for initialization of attributes)
  <== $X \in$ C_meta, $O \in$ C_target_whole, X.key = O.key

We illustrate the use of such a rule by an example. We extend the definition of the type of *ProteinModel_meta* as given in Section 3.2 , since in practice the name of the PDB file does not coincide with the name of the protein. Also one PDB file may contain several protein models for the same protein. Thus the extended type of *ProteinModel_meta* is as follows.

$\text{type}_{\text{ProteinModel\_meta}}$ =
(proteinname:STRING, modelname:STRING, pdbfile:STRING, modelnumber:INT)

A corresponding sample entry is

ProteinModel_meta(”1atr”,”m1”,”1atr.pdb”,1)

A rule that is applied to this entry is

$M \in$ ProteinModel, M.protein = P, $M \in$ P.models,
M.name = PM.modelname
<== PM $\in$ ProteinModel_meta, P $\in$ Protein,
P.proteinname = PM.proteinname

We omit the further type definitions and rules, that are required to describe the transformation of the atom model contents identified by *modelnumber* contained in the file *pdbfile*. Those are straightforward generalizations of the atomic transformation rules given in Section 3.2.

(4) *Establishing relationships between objects:* When relationships are not hierarchically organized it is in general not possible to establish the relationship always at object creation time. For example, the same ligand can be part of many complex structures. Then the explicit creation of a relationship is required after the ligand is created as an object. We give the general pattern for establishing a part-relationship

P.whole = O, P $\in$ O.parts, *(clauses for initialization of attributes)*
<== X $\in$ C_meta, O $\in$ C_target,
X.wholekey = O.wholekey, P $\in$ C_target_part,
X.partkey = P.partkey

It is important to note that the semantics of the rules can be easily communicated to the biological user in an informal manner, e.g.

”The entry *Protein_meta(”1atr”)* leads to the creation of a protein with name *1atr*”

”The entry *ProteinModel_meta(”1atr”, ”model1”, ”1atr.pdb”, 1)* leads to the insertion of a new model with name *model1* for protein *1atr* which is derived from the first model that can be identified in the PDB–file *1atr.pdb*”

The second example is a rule where the corresponding metadata transformation leads to the subsequent execution of an atomic transformation, in this case the transformation of the three-dimensional protein model. This semantics is also easily understood by the user.

3.5 Implementation of data transformations

Given a transformation rule, there exists a source database in form of a record-oriented file, either original files from biomolecular databases or a file prepared by the user containing metadata, and a target database that has to be populated according to the rule. The target database is implemented with an object-oriented database application with the database schema described in section 4.2. We describe by means of an example how in such a case the transformation rule can be implemented. Take the rule

P $\in$ Protein, P.proteinname = Z.proteinname
<== Z $\in$ Protein_meta

introduced in Section 3.4, case (2). Then the implementation of the rule consists of three steps.

(1) Identify the pattern given on the right hand side of the rule. This is done by parsing, if the data resides in a file-based database, or by executing a method resp. query, if the data resides in the object-oriented target database.

(2) Bind the variables occurring in the pattern on the right hand side of the rule.

(3) Execute the transformation actions in order to satisfy the constraints specified on the left hand side of the rule. This is done be sending appropriate messages to the object-oriented DBMS.

In addition we implement the intra database constraints, for example

P = $\text{Mk}_{\text{Protein}}$(P.proteinname) <== P $\in$ Protein

The constraints are checked within the methods executed in the OODBMS. We give now the complete algorithm for the example, similarly as it is actually implemented in our system. The algorithm is available as database method of a class *MetaDB.*

```
MetaDB::transform();
{
while( not eof() )
line := next_line();
/* read next line from file */
case
    line->head() == ”Protein”;
    /* step 1, pattern identified */
    {
    name == line->column(1);
    /* step 2, variable binding
    for Z.proteinname*/
    p := Protein->newWithName(name);
    /* step 3, executing actions
    and checking consistency constraints */
    }
... /* other cases treated similarly*/
end; /* case */
end; /* while */
}
```

In the implementation we have used the method *newWithName()* of class *Protein*. This method performs the required actions and constraint checking. Its implementation is given as follows.

```
Protein::newWithName(n: STRING) : Protein;
{
if(not (Protein -> existsWithName(n)))
/* constraint checking */
{
    p := Protein -> new();
    /* action for P ∈ Protein */
    p.name := n;
    /* action for
    P.proteinname = Z.proteinname */
    return p;
}
else raise(error);
}
```

As a guideline the actions of more complex rules involving actions on different database objects are grouped, such that each group of actions can be realized as a database method of a particular target class. A more detailed architecture of the required software components is given in the next section.

In our implementation the entries are processed in the linear order that is given in the metadatabase by the user. This requires that the entries are ordered in a way that their interdependencies are observed. For example, an entry for inserting an organism into the attributes of a protein object requires that the entry for the creation of the corresponding protein object has to be processed first. Such an ordering appears to be natural for the user and is provided currently without difficulty by him. In the future a component that consistently orders the entries seems to be a useful supplement to the current implementation.

3.6 The design of the integration software layer

As already mentioned in the previous section the data transformations corresponding to rules, that apply to the metadatabase entries, are realized as database methods of a class *MetaDB*. This is also the case for the atomic transformation rules. For each file type that can be integrated into the data warehouse a corresponding class exists in the database schema. For example there exists a class *PDBfile* that provides different parsing methods for PDB files. This establishes in the object-oriented database schema in addition to the application layer, that contains the application schema introduced in section 4.2, an *integration layer.*

The integration layer also allows to keep a history of which information of external databases has been transformed in the database and to track back how the information in the target database has been generated from the source databases (see req (4)). For this purpose corresponding links are supported from classes of the integration layer to the classes of the application layer in the target database and vice versa. This allows to use the original information in the source database files, when required. For example many visualization tools for proteins are able to interpret PDB files directly. In this case we can integrate such a visualization tool easily with our target database schema. Keeping these backward references in the application layer directly is problematic, since there is no one-to-one correspondence between objects in the source and target databases.

In Figure 1 the software architecture is depicted. We distinguish three layers: (1) the external file database layer, (2) the integration layer and (3) the application layer. The layers are represented in an object–oriented diagram. Nodes represent objects and corresponding class objects, arrows indicate method calls, pointing to the receiver objects. Prefixed integers of method calls indicate the order of method invocation. The diagram shows the actions that take place when interpreting the two metadatabase entries *Protein_meta(”1atr”)* and *ProteinModel_meta(”1atr”, ”m1”, ”1atr.pdb”, 1).*

First we create a named reference object for the metadatabase and start the hard coded rule interpreter (method calls 1,2,3). The first metadatabase entry *Protein_meta(”1atr”)* triggers the dispatching of methods 4,5, that create a uniquely named protein object in the target database. The second metadatabase entry *ProteinModel_meta(”1atr”, ”m1”, ”1atr.pdb”, 1)* causes first the creation of a model object *m1* that is linked by its unique name to the protein object *1atr* (method calls 6,7). The next action is the atomic transformation of the atom data from the PDB–file into the target database. Method calls 9,10 create a reference object for the PDB–file and method call 11 parses the PDB–file for *m1*. Method call 12 writes the parsed data into the model object.

4 Implementation of the data warehouse

In this section we give some remarks on the concrete implementation, including the VODAK database management system, the application schema, querying and visualization techniques and finally quantitative features of the system.

4.1 The OODBMS VODAK

The protein-ligand database ReLiBase is implemented on top of the object-oriented database management system VODAK release 4.0 [24] developed at GMD-IPSI. VODAK provides full database management system functionality, including schema compiler, transaction management, object management, data dictionary, communication management and query processing. It runs on top of the commercial system ObjectStore, which is used for low-level storage.

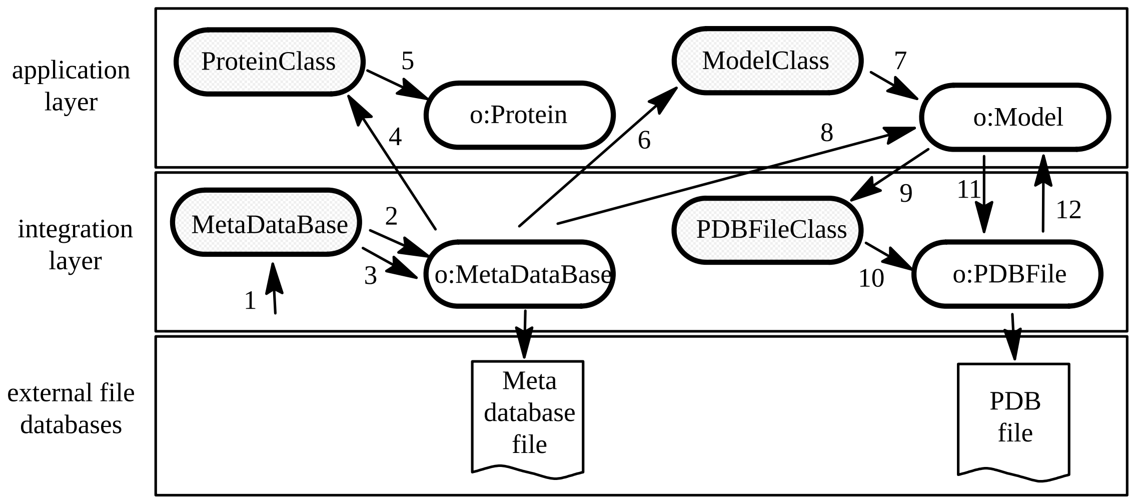

Figure 10: Processing of the metadatabase
1:newWithName(”load”), 2:setName(”load”), 3:parse(), 4:newWithName(”1atr”), 5:setName(”1atr”), 6:newWithName(”1atr,m1”), 7:linkToProtein(p), 8:setModel(”1atr.pdb”,1), 9:newWithName(”1atr.pdb”), 10:setName(”1atr.pdb”), 11:setAtoms(as), 12:parseModel(1)

Different features make VODAK particularly attractive for our purposes. The object-oriented data model VML provides a metaclass concept for tailoring the data model to specific requirements, like database integration [17]. Thus semantic concepts that are characteristic for an application domain can be provided as new data modelling primitives extending the core data model. The declarative OQL compatible query language allows the usage of method calls and path expressions within queries, and thus enables flexible and easy access to the database. A powerful, extensible query optimization module, based on algebraic optimization techniques, allows to incorporate application-specific optimization strategies, which are particularly important when complex calculations or expensive access operations to external databases are involved in declarative queries [2].

4.2 Implementation of the application schema

The schema provides a uniform logical view on the underlying data resources as well as on integrated tools. An overview of the schema is given in Figure 2. The figure also illustrates how object-oriented data modelling primitives allow a direct representation of the complex interconnection between the different biological data types.

For performance reasons we had to provide for certain data types, for example the three-dimensional models of proteins, specialized physical implementations, where the logical object-oriented view of the data and the physical representation differ. In the spirit of data independence these specialized implementations can be accessed through abstract interfaces in a way such that the user is not aware of the underlying specialized mechanisms.

The schema provides for some of the application specific data structures specialized index structures, which are accessible by dedicated methods. Thus they can either be directly used or are potential alternative access mechanisms the query optimizer can choose from. Other methods encapsulate the access to external systems, like visualization tools or external databases, that are not physically integrated in the warehouse.

4.3 Query processing

Query optimization plays a central role for the efficient usage of the data warehouse. For example, in some cases we do not migrate data physically into the warehouse. In those cases only access mechanisms for external databases are provided. Those databases often provide alternative access paths, that allow for more efficient and adequate processing of specialized requests. To exploit these alternative access paths, we make the query optimizer aware of the semantics of these.

Another aspect of query optimization is to minimize the of expensive methods. For some queries the method execution costs are the dominating factor, and optimization strategies have to avoid too frequent method calls. Thus also knowledge on the execution costs of methods is made explicit in our system.

Application specific index structures, e.g. for efficient access to spatial data, and special purpose physical data representations play an important role in the warehouse. The

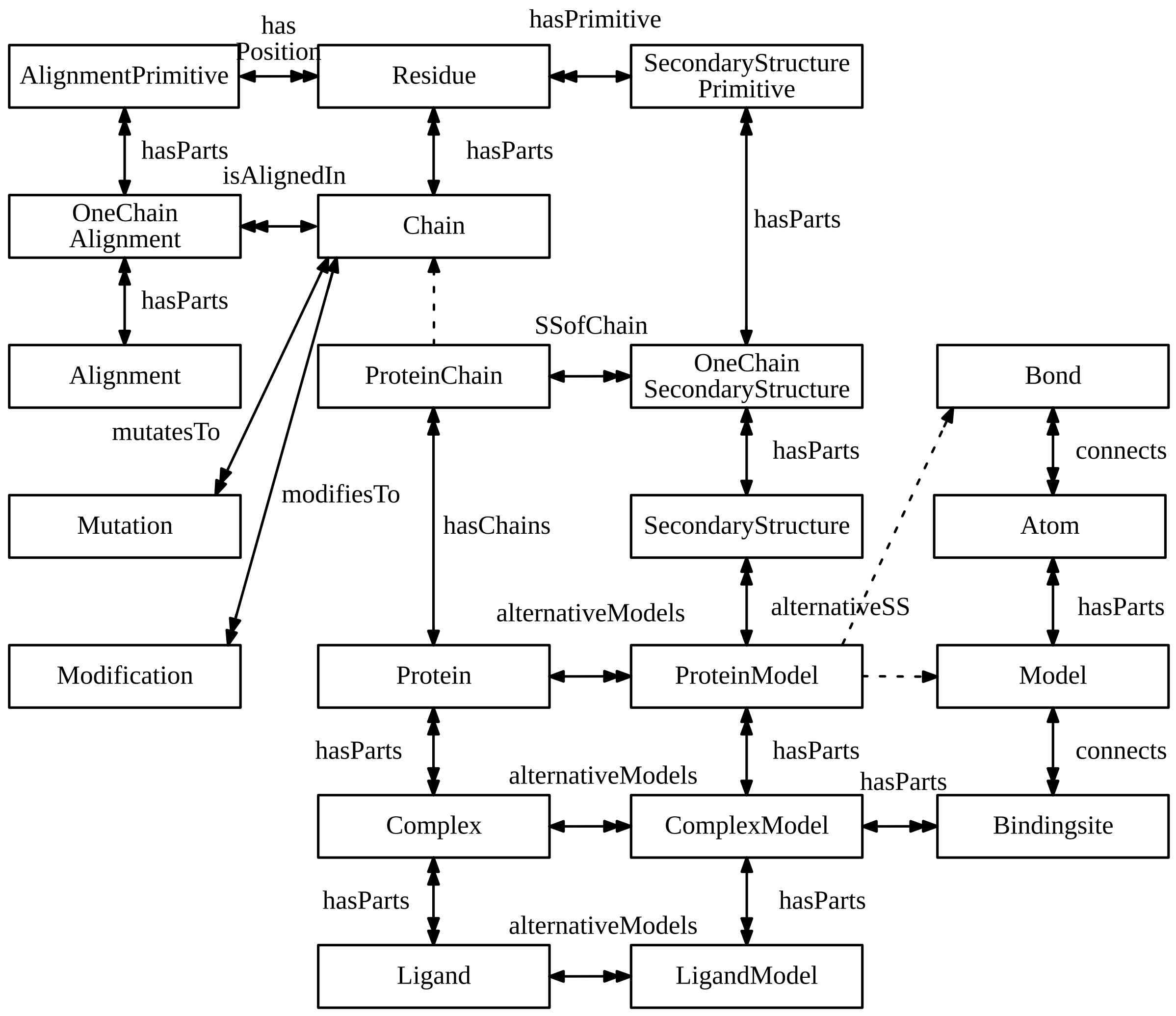

Figure 11: The conceptual database schema for the data warehouse ReLiBase.

correct usage of index structures and specialized access mechanisms can be a difficult task for the user. Knowledge on the semantics of these mechanisms is thus made available to the query optimizer, such that they can be hidden from the user in ad-hoc querying.

Summarizing, it is crucial to have an open query processing component that supports extensibility with regard to query transformations and cost models. In VODAK we realize this through a query processing module that follows an extensible design [2][3].

4.4 Visualization

Biological data visualization tools are an important means to understand the structure and function of molecules in detail. From another perspective also the object-oriented data model is a paradigm that is predestined to be accessed in a visual way.

Thus we have integrated several visualization tools with our database. To support data inspection, browsing and navigation we provide a generic HTML-based object viewer, that allows to navigate along object references, to display the objects attributes and to send messages to objects. Queries can be submitted via HTML, either as predefined queries, parametrized queries or, by the more sophisticated user, by directly writing OQL queries according to the ODMG93 standard [9]. Query results are formatted in a way that the results become starting points for navigational access in HTML.

For viewing biomolecular data we have integrated generic 3D viewers for VRML, as well as specific biomolecular viewers like RASMOL or Whatif [26]. Specialized viewers operate only on a particular kind of data, for example proteins. As we are integrating data from different sources viewers for the integrated data, e.g. for presenting complexes, may not be available. In such cases generic viewers like VRML-based ones are used. Another interesting aspect is to use visualized objects as starting points for consecutive navigation. For example one gets the properties of an atom from the database by clicking at this atom in a visualization tool.

4.5 Experiences

The ReLiBase system is implemented and running. Currently we are evaluating the system together with the users. As a result extensions and improvements of the system will be developed and new requirements will be derived. We give a short list of quantitative features to illustrate the order of magnitudes we are dealing with.

database instances	
instances in the metadatabase	200.000
instances in the target database (physical DBMS objects)	400.000
protein–ligand complexes (target database)	3500
database generation	
populating a complete warehouse	4 days
indexing the complete warehouse	1 week
size of the warehouse	
without index structures	900 MBytes
with index structures	1.5 GByte
size of the source databases	600 Mbytes
duration of a typical scan query	
over all complexes (cold)	8 sec
over all complexes (hot)	4 sec
complete scan of the warehouse	10 hours

All of the numbers are approximate as they can depend on configuration and system usage. They are taken from a snapshot of the current system version and thus can be considered as typical values. The system is running on a SGI Challenge 800 L, with 256 MBytes Main Memory using one processor at 200Mhz.

The duration of query evaluation is often dominated by the algorithmic cost for executing specialized analysis algorithms. For example, evaluating a query that requires to execute a substructure test for the ligand of each complex takes approximately 2 hours, since each single evaluation, which performs basically subgraph testing, takes about 2 sec. for each of the 3500 complexes in the database.

5 Conclusion

The methodology and system presented in this paper is currently used to build a data warehouse for data analysis purposes in drug design. The development of the approach was driven by the requirements that occurred in the cooperation with industrial users. The system provides integrated and efficient access to an extensively analyzed, corrected and enriched database. The advantage of the methodology is the separation of the tasks of data analysis, that require substantial biomolecular knowledge, from the technical details of data transformation.

Further work is driven by requirements that occur in the practical use of the system. In order to keep the system scalable for the ever growing amounts of data, it appears mandatory to extend the techniques such that they allow for incremental loading and indexing of the data. Due to VODAK's extensibility of the data model and query processing module the system will grow with regard to built-in intelligence in handling complex algorithms and accessing databases that are not physically integrated in the warehouse.

Data warehousing is receiving growing attention in the computer science literature [28]. As most activities in this direction are quite recent no solutions for our concrete database integration problems have been readily available. We mention some relevant work and contains potentially interesting ideas for the methodological construction of complexly structured data warehouses. A mediatory approach that focuses on rule-based specifications for data warehousing, that are comparable to ours, is reported in [30]. Other relevant work is found on object migration [22][21], database transformation [19][8], mediatory systems [29] or the access to file-based databases from within object-oriented databases [4].

6 References

[99] Aberer, K.: ”The Use of Object–Oriented Data Models for Biomolecular Databases”, Proceedings of the Conference on Object–Oriented Computing in the Natural Sciences (OOCNS) ’94 , Heidelberg, 1995.

[100] Aberer, K., Fischer, G.: ”Semantic Query Optimization for Methods in Object–Oriented Database Systems”, International Conference on Data Engineering (DE) ’95, Taipei, Taiwan, p 70–79, 1995.

[101] Aberer, K., Hemm, K.: ”Semantic Optimization of Biomolecular Queries in Object–oriented Database Systems”, Meeting on the Interconnection of Molecular Biology Databases (MIMBD) ’95, Cambridge, UK, WWW page http://este.darmstadt.gmd.de:5000/~aberer/MIMBD95.html, 1995.

[102] Abiteboul, S., Cluet, S., Milo, T.: ”Querying and Updating the File”, Proc. of Int. Conf. on VLDB, p. 73–84, 1993.

[103] Bairoch, A., Boeckmann, B.: ”The SWISS-PROT Protein Sequence Data Bank”, Nucleic Acids Res., 19 (Sequences Suppl.), p 2247 – 2249, 1991.

[104] Bernstein, F.C., Koetzle, T.F., Williams, G.J.B., Meyer, E.F. Jr., Brice, M.D., Rodgers, J.R., Kennard, O., Shimanouchi, T., Tasuni, T.: ”The Protein Data Bank: a computer based archival file for macromolecular structures”, J. Mol. Biol. 112, p 535 – 542, 1977.

[105] Buneman, P., Davidson, S.B., Hart, K., Overton, C.: ”A Data Transformation system for Biological Data Sources”, Proceedings of the 21st VLDB Conference, Zurich, Switzerland, 1995.

[106] Cabibbo, L.: “On the Power of Stratified Logic Programs with Value Invention for Expressing Database Transformations”, ICDT, p 208–221, 1995.

[107] Cattell, R. G. G. (Ed.): ”Object Databases: The ODMG-93 Standard”, Release 1.1, Morgan Kaufmann, San Francisco, 1994.

[108] Durbin, R., Thierry–Mieg, J.: ”The ACEDB Genome Database”, WWW page, http://probe.nalusda.gov:8000/acedocs/dkfz.html.

[109] Etzold, T., Argos, P.: ”SRS an indexing and retrieval tool for flat file data libraries.”, Comput. Appl. Biosci. 9:49–57, 1993.

[110] Gardarin, G., Gannouni, S., Beatrice Finance, B., Fankhauser, P., Klas, W., Pastre, D., Legoff, R., Ramfos, A.: IRO–DB: A Distributed System Federating Object and Relational Databases. In: Omran Bukhres and Ahmed K. Elmagarmid, editors, Object Oriented Multidatabase Systems, Prentice Hall, July 1994.

[111] Goodman, N.: ”An Object-Oriented DBMS War Story: Developing a Genome Mapping Database in C++”, in Kim W., ed.: ”Modern Database Systems”. ACM Press, New York, p 216 – 237, 1995.

[112] Hemm, K., Aberer, K., Hendlich, M.: ”Constituting a Receptor–Ligand Database from Quality–Enriched Data”, Proceedings of the International Conference on Intelligent Systems in Molecular Biology (ISMB) ’95, Cambridge, UK, 1995.

[113] Inmon, W.H., Kelley, C.: ”Rdb/VMS, Developing the Data Warehouse”, QED Publishing Group, 1993.

[114] Karp, P. (Ed.), Final report on the 1994 Meeting on Interconnection of Molecular Biology Databases, WWW page http://www.ai.sri.com/people/pkarp/mimbd.html, Stanford University, August 9–12, 1994.

[115] Klas, W., Fankhauser, P., Muth, P., Rakow, T., Neuhold, E.J.: ”Database Integration Using the Open Object-Oriented Database System VODAK”, in: Object-Oriented Multidatabase Systems, O. Nukhres, A.K. Elmagarmis Eds., Prentice Hall, 1995.

[116] Kemp G.J.L., Jiao Z., Gray P.M.D., Fothergill J.E.: “Combining Computation with Database Access in Biomolecular Computing”, ADB 94, Vadstena, Sweden, p 317–335, 1994.

[117] Kosky, A., Davidson, S., Buneman, P.: ”Semantics of Database Transformations”, Technial Report MS–CIS–95–25, University of Pennsylvania, 1995.

[118] Nishikawa,K., Ishino,S., Takenaka,H., Norioka,N., Hirai,T., Yao,T. and Seto,Y.: ”Constructing a protein mutant database”, Protein Engng, 7, 733, 1994.

[119] Qing Li, Guozhu Dong: A Framework for Object Migration in Object–Oriented Databases. DKE 13(3), p 221–242, 1994.

[120] Radeke, E., Scholl, M.: ”Federation and Stepwise Reduction of Database Systems”, ADB 94 Vadstena, Sweden, p 381–399, 1994.

[121] Ritter, O., Kocab, P., Senger, M., Wolf, D., Suhai, S.: “Prototype Implementation of the Integrated Genomic Database”, Computers and Biomedical Research, 27, p 97–115, 1994

[122] VODAK V 4.0, User Manual, GMD technical report No. 910, 1995.

[123] Sheth, A.P., Larson, J.A., ”Federated Database Systems for Managing Distributed, Hetereogenous, and Autonomous Databases”, ACM Computing Surveys, vol. 22, no. 3, p 183 – 236, 1990.

[124] Vriend, G.: ”WHAT IF: A molecular modeling and drug design program.”, J. Mol. Graph. 8, 52–56, 1990.

[125] Wäsch, J., Aberer, K.: ”Flexible Design and Efficient Implementation of A Hypermedia Document Database System by Tailoring Semantic Relationships”, Proceedings of the IFIP WG2.6 sixth Working Conference on Database Semantics (DS–6) Atlanta, Georgia, USA, 1995.

[126] Widom, J., (Ed.), Data Engineering Bulletin 18(2), Special Issue on Materialized Views and Data Warehousing, 1995.

[127] Wiederhold, G.: ”Mediators in the architecture of futurre information systems”, IEEE Computer, 25:38–49, 1992.

[128] Zhou G., Hull R., King R., Franchitti J.C.: “Data Integration and Warehousing Using H2O”, Data Engineering Bulletin 18(2), pp. 29–40, 1995.