Design and Implementation of a Biodiversity Information Management System

Srikanta B. J.
Database Systems Lab, SERC
Indian Institute of Science
Bangalore 560012, India
srikanta@dsl.serc.iisc.ernet.in

Jayant R. Haritsa*
Database Systems Lab, SERC
Indian Institute of Science
Bangalore 560012, India
haritsa@dsl.serc.iisc.ernet.in

ABSTRACT

Over the last decade, there has been a phenomenal growth in the area of *bio-diversity* studies, largely motivated by its economic and humanitarian potential. We present here the design and implementation of BODHI, a bio-diversity information management system, providing a generic framework for datamanagement needs of the domain. The system efficiently integrates diverse datatypes, from genetic to ecological details, incorporating latest techniques in data management. The system has been designed to seamlessly integrate with the internet and supports flexible data interchange between repositories. The scalability and lowcost of the system make it suitable for both large and small bio-diversity data collection and study efforts.

1. INTRODUCTION

Over the last decade, there has been a phenomenal growth in the area of *plant bio-diversity* studies, largely motivated by the economic and humanitarian potential that underlies the understanding of plant dynamics – in fact, a new area called "bio-prospecting" has arisen, whose focus is solely on sifting through bio-diversity data to locate potentially profitable biological sources.

A major hurdle faced by bio-diversity scientists is the effective management and access of the large amounts and varied types of data that arise in their studies, ranging from micro-level biological information (such as genetic makeup of organisms and plants) to macro-level information including ecological and habitat characteristics of species. Six years ago, we made a first attempt to address the above problem in the *Oshadhi* [34] project, which developed an object-based database system for efficiently handling plant taxonomies. However, it had only a rudimentary spatial-data component and a grossly simplified query processor/optimizer. Further, during the interim period, there have been a variety of new technologies that have arisen both on the biological and on the database fronts. These include the Internet/Web, XML, sophisticated spatial indexing schemes, genome sequencing algorithms, and semistructured data-handling techniques, all of which can be usefully incorporated in biodiversity information systems.

In light of the above, over the last year, with financial support from the Department of Bio-technology, Government of India, and in collaboration with the biologists of the Centre for Ecological Sciences at our institute, we have been developing the next generation biodiversity information system, called **BODHI** (Biodiversity Object Database arcHItecture).[1] BODHI is an object-oriented database system built around the SHORE kernel [6] which incorporates the latest algorithms and data handling structures, provides integration with the Internet for data dissemination, and handles the needs of bio-diversity researchers with diverse focus areas. In this paper, we describe our experiences in the design and implementation of the BODHI prototype.

Organization

The rest of the paper is organized as follows: We briefly describe the design goals in Section 2, and present BODHI's architecture in Section 3. The implementation details are discussed in Section 4, while query processing is addressed in Section 5. The user interface is highlighted in Section 6. The current status of BODHI system is outlined in Section 7, and related work is compared in Section 8. Finally, we summarize our work in Section 9.

2. DESIGN GOALS

In this section, we highlight the main features that would be desirable in a bio-diversity information system.

Handling of Complex Data Types: Plant bio-diversity data can be broadly classified into the following three categories: (1) **Taxonomy Data.** This is data about

*Also affiliated with the Dept. of Computer Science & Automation, Indian Institute of Science. Part of this work was done while on sabbatical at Lucent Bell Labs, 600 Mountain Avenue, Murray Hill, NJ 07974, USA.

[1]Gautama Buddha gained enlightenment under the Bodhi tree.

ADVANCES IN DATA MANAGEMENT 2000
Krithi Ramamritham, T.M. Vijayaraman (Editors)
Tata McGraw-Hill Publishing Company Ltd.

the relationships between species based on their characteristics. In turn, this consists of *phenetic relationships* (based on comparison of physical characteristics or *phenotype*) and *phylogenetic relationships* (based on evolutionary theory)[28]. The various characteristics on which these relationships depend on may vary in time due to discovery of a new class of characteristics, corrections to previously recorded characteristics, etc.; (2) **Geo-spatial Data.** The study of ecology of species involves recording the geographical and geological features of their habitats, water-bodies, artificial structures like highways which might affect the ecology, etc. These are represented on a map of the region and have to be handled as spatial data by the database; (3) **Bio-molecular Data.** The genetic makeup of species is becoming increasingly important with a large number of genome sequencing projects working on organisms and plants. Bio-prospectors look for indigenous sources of medicines, pesticides and other useful extracts. Such data can be discovered from the biomolecular and genetic composition of species.

These datatypes have complex and deeply-nested relationships within and between themselves. Further, they may involve sophisticated structures such as sequences and sets.

Molecular Pattern Discovery: The molecules that are of interest in bio-diversity are DNA and Proteins. DNA is represented as a long sequence based on a four nucleotide alphabet. There are regions in DNA sequence, called *exons*, which contain genetic code for the synthesis of Proteins. The proteins are long chains of 20 amino acids. Each protein is characterized by the amino acid patterns it has, and is responsible for various functionalities in a cell which determine the characteristics of the organism or plant.

The similarity between two genetic sequences is a measure of their functional similarity. Analysis of DNA and Protein sequences from different sources gives important clues about the structure and function of Proteins, evolutionary relationships between organisms, and helps in discovering drug targets.

There are a number of algorithms for performing the similarity search over genetic sequences. Researchers and bio-prospectors frequently search the database using these algorithms to locate gene sequences of interest. However, the implementation of these algorithms is typically external to the database, making them relatively slow. It therefore appears attractive to consider the possibility of integrating these algorithms in the database engine. [2]

Internet Access: As with all other scientific communities, the bio-diversity community relies on timely knowledge dissemination. Therefore, supporting access through the Internet is vital for maximizing the utility of the information stored in the database.

[2]This observation is gaining currency in the commercial database arena as well, as exemplified by IBM's provision of homology searching as UDFs in DB2.

XML Compliant: Typically, bio-diversity data is collected and managed by individual research institutions and commercial enterprises autonomously. In order to provide larger scope of data availability, it is necessary that such localized and autonomous data repositories be able to exchange data. The current state of information exchange amongst various bio-diversity data repositories is not very satisfactory[29]. However, with the advent of XML, many research groups are proposing DTDs in individual fields of ecology and genetics[2, 3] . The bio-diversity information system should support these DTDs for handling data over heterogenous set of repositories.

Visual Interface: It is imperative to have a good visualization interface for the results produced by the system since (a) the end-users are biologists, and (b) the results could range from simple text to multidimensional spatial objects. The database system must support easy integration of a visual interface.

Low Cost: Most of the research in bio-diversity, at least in India, is done by small teams of researchers who work within low budgets and are unable to afford high-cost data repository systems. Therefore, solutions that are completely or largely based on public-domain freeware are essential for these groups.

3. DESIGN DESCRIPTION

In this section, we present the design of the BODHI system intended to efficiently achieve the goals described in the previous section. The overall architecture is pictorially shown in Figure 1. The back-end micro-kernel is the SHORE [6] object-based storage manager, which provides the basic database features such as device and storage management, transaction processing, logging and recovery management. Above this are three modules, *Object Services*, *Spatial Services* and *Sequence Services*, which provide the core functionalities of the BODHI system – these modules are described in detail in the remainder of this section. The three services are integrated through the *Query Processor* which interacts with the service modules and the SHORE server to optimize and process the queries on the database. Clients submit queries, over the Internet, in the form of OQL [7] command strings – the *Client Interface Framework* receives these queries and returns results in the formats requested by the clients.

3.1 Object Services

In BODHI, the object oriented paradigm is used to achieve the following features necessary for bio-diversity data:

- Support multiple data primitives through the use of type libraries at the database layer. (The spatial and genome sequence data primitives are provided using this facility.)
- Representation of complex relationships such as sets, sequences and bags.
- Build new types through inheritance and aggregation of previously defined non-primitive types.

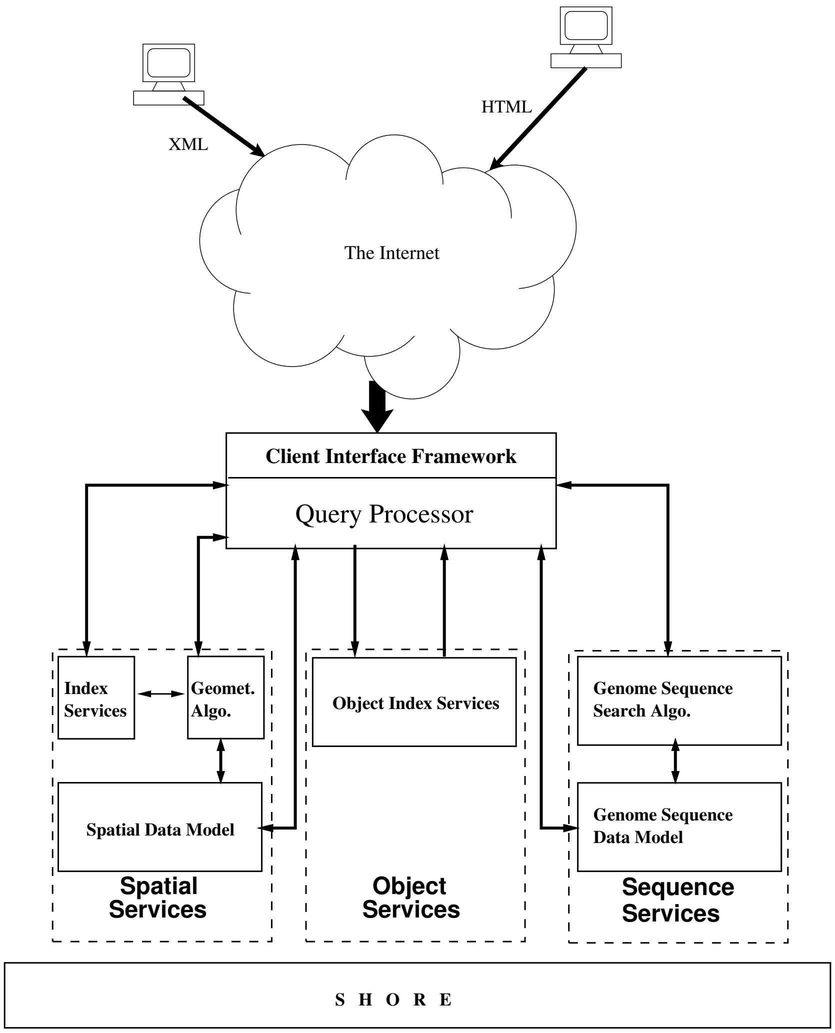

Figure 1: Schematic of Architecture of BODHI

The data modeling language of BODHI extends the standard ODL[7] by introducing new primitives for modeling spatial and sequence data. These primitives can be used in conjunction with standard primitives provided by ODL and various relationships between objects can be easily modeled. The schema definition supports two types of relationship-paths over objects, namely, Inheritance hierarchies and Object Relationship paths. The queries over the database consist of both value based queries and also on the context of the object in the relationship graph and the position of its type in the associated class hierarchy.

3.1.1 Indexing the Inheritance Hierarchies

Based on the context, the scope of queries over inheritance hierarchies in the bio-diversity domain can be either limited to the immediate objects of the predicate class (i.e. single-class query) or can be extended to include all objects of the sub-hierarchy rooted at the predicate class (i.e. class-hierarchy query). For example, with reference to the schema of Figure 2, which is a (partial) class diagram of the plant taxonomy model,

Give names of all **PlantSpecies** *associated with a* **GeoRegion**

We have two possible semantics for this query, (i) search in the complete inheritance hierarchy rooted at *PlantSpecies* and return objects of type *PlantSpecies* and *MedicinalPlants* associated with the given *GeoRegion*, (ii) search objects of only *PlantSpecies* type associated with given *GeoRegion*, without searching for *MedicinalPlants*.

To efficiently support both types of queries, the *Multi-key Type Index*(MT-index)[26] approach is used in BODHI. The basic idea behind MT-index is a mapping algorithm, called *Linearization Algorithm*, which maps type hierarchies to linearly ordered attribute domains in such a way that each sub-hierarchy is represented by an interval of this domain. Using this algorithm, MT-index incorporates the type

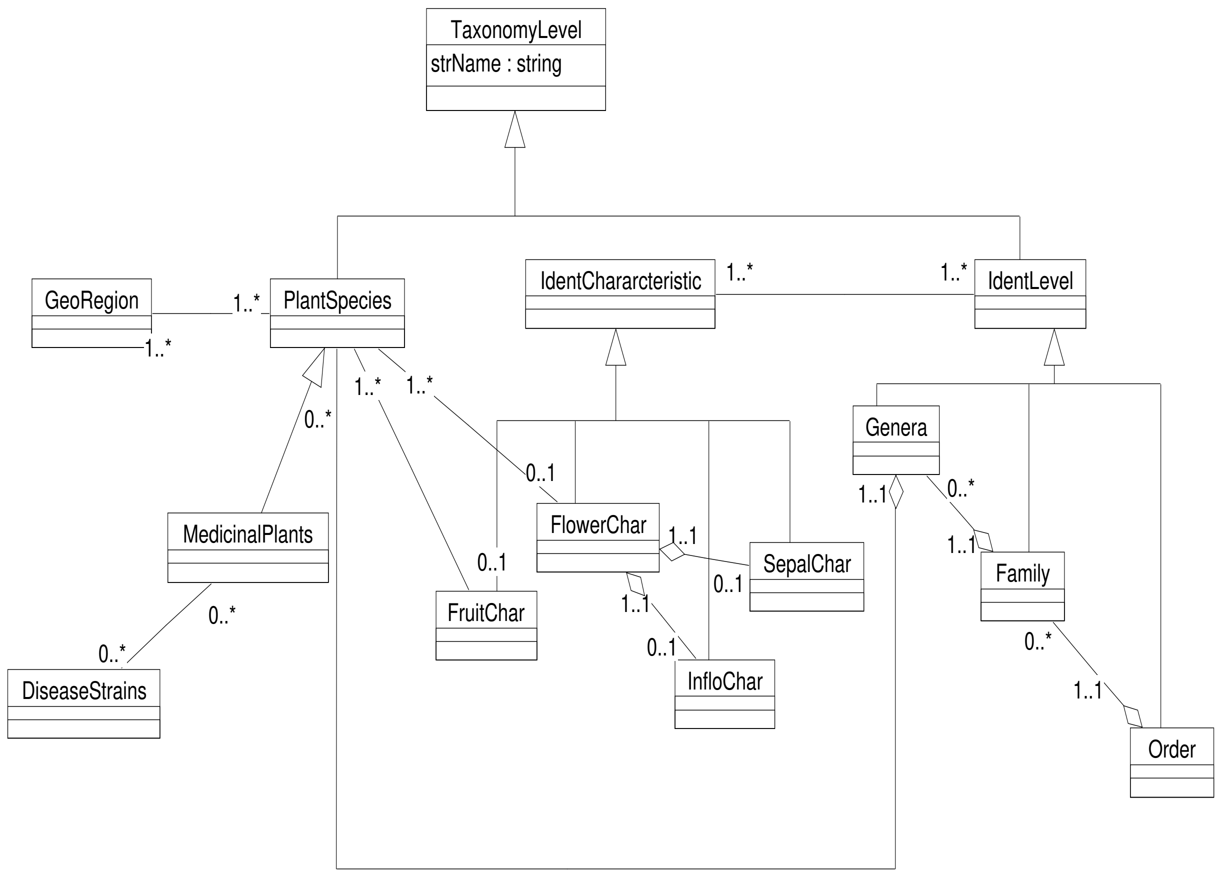

Figure 2: Partial Schema of Plant Taxonomy Database.

hierarchy structure into a standard multi-attribute search structure, with the hierarchy mapped onto one of the attribute domains (type domain). The result is an index on $K+1$ keys, K indexed object attributes and one additional key representing the type. This scheme not only supports single-class and class-hierarchy queries, but also makes answering multi-attribute queries straight forward. Further, the implementation of MT-index can use any of the multi-dimensional indexing schemes (like R^*-trees, supported by SHORE), which simplifies the implementation overhead.

1. Using a combination of two hierarchy indexing approaches, MT-index can answer both single and class hierarchy queries with equal efficiency.

2. By extending the number of dimensions, queries over multiple attributes can be easily answered without any time overhead.

3.1.2 Indexing the Relationship Paths

Referring again to the schema of Fig 2, we can see that the relationship hierarchy between objects in bio-diversity database schema can be arbitrarily deep. Further, it is possible to have recursive relationships, for example, the *Predator-Prey* relationship among Species. Queries over such relationship graphs can have either the ancestor class or the nested class, as the predicate class. To illustrate, consider the following pair of queries:

- *Identify the* **PlantSpecies** *based on one or more of its* **IdentCharacteristics**.
  Here, we need to scan for object relationship path(s) culminating at the specified characteristic, and then locate the species that form the root(s) of that path(s). Such queries are called TP (Target-Predicate) queries, following the terminology of [22].

- *Retrieve all* **IdentCharacteristics** *of a given* **PlantSpecies**.
  Here, the predicate classes are ancestor classes (PlantSpecies) and the target classes are nested classes (IdentCharacteristics). These types of queries are called PT (Predicate-Target) queries in [22].

To efficiently handle both PT and TP queries, BODHI implements the *Path Dictionary Index*(PD-Index)[22] approach. The PD-Index consists of three parts: the *path dictionary* which supports the efficient traversal of the path,

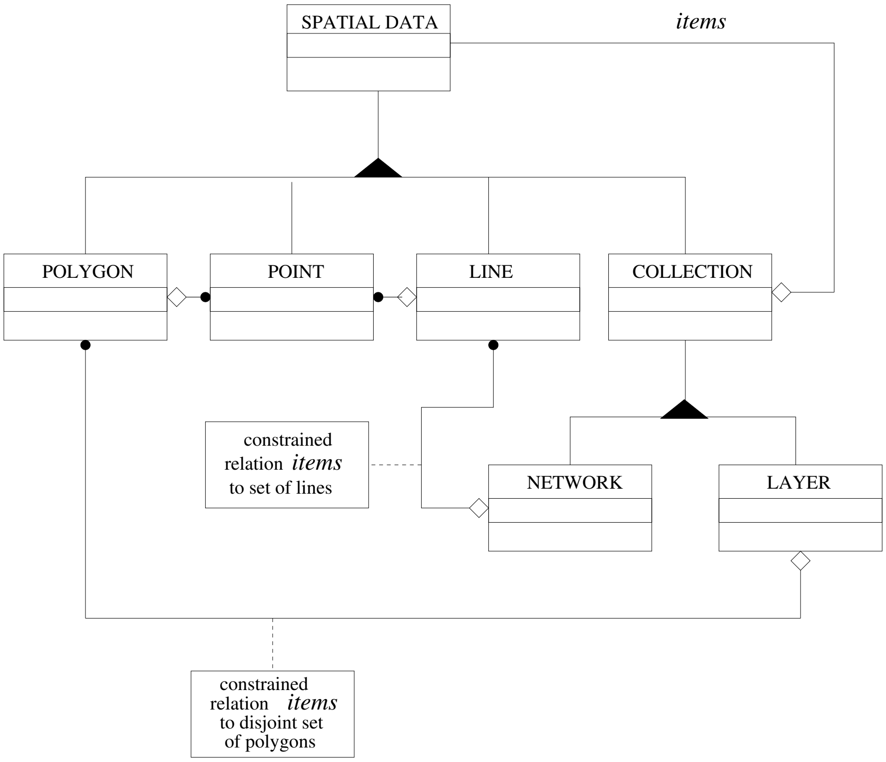

Figure 3: Class diagram of Spatial Data model in BODHI

and the *identity index* and *attribute index* which support associative search. The identity index and attribute index are built on top of the path dictionary.

Conceptually, the path dictionary extracts the compound objects, without the primitive attributes, to represent the connections between these objects. Since primitive attribute values are not stored in the path dictionary, it is much faster to traverse the nodes. In order to support associative search based on attribute values, PD-Index provides attribute indexes which can be built for each attribute frequently queried on. When OID of the object is given, the path information is obtained using the identity index built over the path dictionary.

The PD-index provides the following advantages:

- Since both forward and backward traversals are supported by the path dictionary, it is possible to evaluate both PT and TP queries with equal efficiency.
- Since intermediate objects need not be traversed in deep aggregations, lot of savings in retrieval time can be achieved.
- Identity index reduces the update overhead by directly pointing the path dictionary locations where update is to be done.

A limitation of the PD-index, as per its implementation description in [22], is that it only handles 1:N relationships or 1:1 relationships. However, a typical schema of bio-diversity database consists of aggregations of N:M cardinality, and structures like sets, bags and sequences in the aggregation path. In order to handle these complex relationships, we have extended the implementation of the PD-Index – the details of our enhancements are in Section 4.

3.2 Spatial Services

Spatial data or geographic data forms a key component of a bio-diversity data repository. The spatial data is frequently queried upon, and a spatial query could involve operations (geometric and topological) which could be extremely costly to evaluate. And more importantly, a large size of macro-level biodiversity information consists of topographic and ecological maps.

In BODHI, we provide *Spatial Data Types* (SDTs) in our data model and query languages (ODL/OQL extended for the purpose), and support efficient spatial indexing and spatial join algorithms.

3.2.1 Spatial Data Types

BODHI provides a set of spatial data primitives to represent single spatial objects like country, state, forest, river etc. as well as to represent spatially interrelated collection of objects such as *"Political map of India"*, which can be modeled as a topologically related collection of polygons representing states.

The standard ODL modeling language has been extended with new keywords to enable users to include these primitives in their schema descriptions.

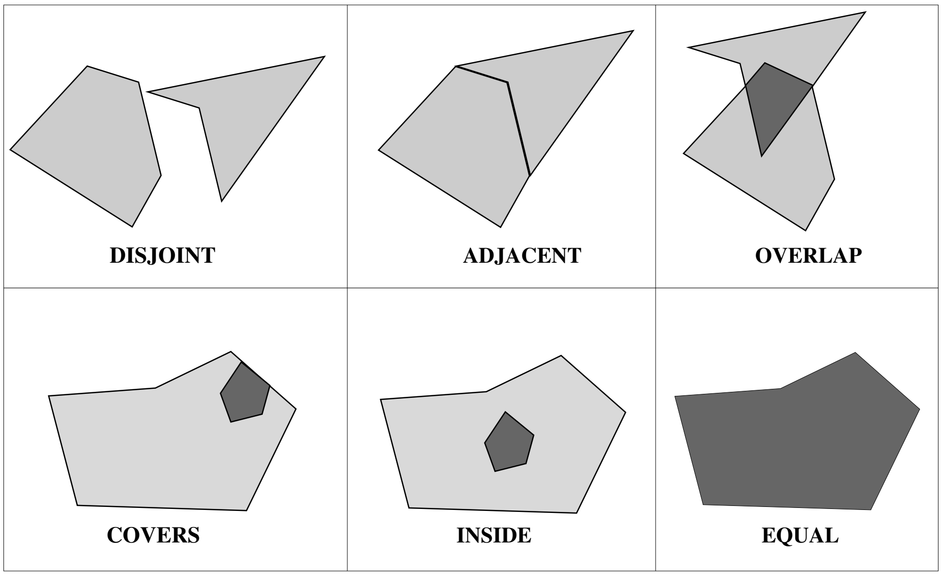

Figure 4: Spatial Relationships in BODHI

The spatial model of BODHI is based on the *Realm/ROSE Algebra*[17], and provides two categories of primitives: *Simple Primitives* and *Compound Primitives*. The simple primitives enable modeling of single objects in space, and includes types for *Point*, *Polyline* and *Polygon*. The compound primitives, on the other hand, are used to model spatially-related collection of objects. There are two compound primitives: *Layer* and *Network*, for modeling collections of *Polygon* and *PolyLine*, respectively.

The fig. 3 gives the class diagram of the Spatial data model of BODHI. As illustrated, the Point forms the basic information entity of the spatial data. Every Polyline is modeled within the type library, using the information about the end points of each segment that forms the Polyline. The Polygons are the collection of single-segment lines.

3.2.2 *Querying on Spatial Objects*

Spatial queries consist of selecting objects which satisfy some spatial relationship(s). There are three classes of such spatial relationships,

- *Topological relationships*, such as *adjacent*, *inside* etc. which are invariant under geometrical transformations like translation, scaling and rotation.
- *Direction relationships*, like *above*, *north-of* etc.
- *Metric relationships*, based on the distance measure between spatial objects. E.g., *"distance < 100"* etc.

Of all the relationships in these three categories, [17] observes that only six relationships, *disjoint*, *in*, *touch*, *equal*, *cover* and *overlap*, which are illustrated in Figure 4, are most important relationships in geo-spatial applications (even though only polygons are used for illustrating the relationships, these are also defined for line, layer and network primitives). The *Geometric Algorithms* part of the Spatial Services of BODHI currently provides these six relationships. These algorithms form the core of behavioural abstraction of the spatial primitives described above.

3.2.3 *Spatial Indexing*

Special indexing structures are required for efficiently accessing spatial objects. While numerous indices have been proposed in the literature (see [12] for a survey), R^*-Trees are extremely popular spatial indexing structures, as they have been designed to achieve better packing of nodes and fewer disk accesses in comparison with many other variants of R-Trees.

A fundamental difficulty in designing of multidimensional index structures stems from the following reason: There is no total ordering that preserves spatial proximity. One way out of this is to find heuristic solutions, that is, to look for total orders preserving spatial proximity to a large extent. The requirement is that if two objects are located close together in multidimensional space, there should be a high probability that they are close together in the total order. One such heuristic is the concept of *space-filling curves*[19] and there have been proposals based on this heuristic (for a survey of these and other spatial access methods, refer to [12]). All the proposed index structures have the following in common: They first partition the multidimensional space with a grid, and then use a curve such as *Peano Curve* or *Hilbert Curve*[10], which visits all the points in the grid only once without crossing itself, to obtain a total ordering of the spatial objects.

BODHI indexes spatial objects using one such proposal, the *Hilbert R-Tree* (HR-Tree) [20], where total ordering is achieved through Hilbert Curve for total ordering. The linear ordering imposed by the space filling curve is used to optimize the structure of the R-Tree. The performance gains have been shown to be more than 25% over R^*-Tree, considering a wide range of workloads[20]. We expect that similar performance gains might be obtained in BODHI as the benchmarks used in [20] closely match the real-life needs.

3.2.4 Spatial Join Strategies

Spatial joins, wherein sets of objects are compared using predicates on their spatial attribute values, are common in bio-diversity applications. For example,
"select all the states in India where the rivers Godavari, Krishna and Kaveri flow".
Obviously, any cartesian product based solution is impractical for spatial joins since they are extremely expensive to evaluate.

However, while spatial index structures can be used for computing the join efficiently, a refinement of the result is required since the index structures approximate each object's contour to its bounding box. So the key ideas for computing spatial joins are the filter and refinement strategy and the use of spatial index structures. If the filter step is based on the bounding boxes, the problem is to determine for two sets of rectangles R, S, all pairs (r,s), r∈R, s∈S, such that r intersects s. There are different policies for join processing depending on the availability of the index on the operands.

- If one operand is represented in a spatial index, then an *index join* or *repeated search join* can be used [24]. This is a classical technique usually used with a B^+-tree index, which can be applied equally well to spatial index structures. Hence, if the "inner" operand is represented in an spatial index supporting rectangle intersection queries one can scan the "outer" operand set; for each object, the bounding box of its SDT attribute is used as a search argument of the index. As a result one again obtains a set of candidate pairs with intersecting rectangles. Repeated search join is especially efficient if the outer set is not too big (e.g., it is the result of a selection from a large set). If both are large, *bb_join* may be more efficient [17] and query processor makes the appropriate choice.
- If both the operands have spatial indexes built, the basic idea is to perform a synchronized traversal of the two index structures so that pairs of cells whose respective partitions cover the same part of space are encountered together. The parallel traversal of R-trees method discussed in [5] is appropriate in BODHI, because it uses any of the R-tree family of index structures and hence can achieve all the benefits of the Hilbert R-tree index structure for join processing also.
- If none of the operands have index, the index of one operand can also be built on the fly, as suggested by [24]. This is considered by the query processor before the next option.
- If none of the operands is represented in a spatial index, a good technique suggested by [17] is to use a rectangle intersection algorithm from computational geometry called *bb_join*. This computes the join using bounding box information of spatial objects rather than the actual extent, making it faster than cartesian product. Later, we perform the filtering over the join result.

The query processor of BODHI chooses the appropriate policy of spatial join computation, from among those listed above, depending on the availability of index over the attributes under query.

3.3 Sequence Services

As we already highlighted in Section 2, the bio-molecular information of various species is growing at a very fast rate. The study of genetic makeup of species in conjunction with macro-level information is becoming an important activity in bio-diversity studies. BODHI facilitates for such integrated study by providing data model and query language support for modeling of biological sequences, and implementing commonly used sequence similarity algorithms over the genome sequence data.

3.3.1 Biological Sequence Primitives

In biology, a bio-molecular sequence is either a *DNA* (Deoxyribonucleic acid) or a *Protein* sequence. These two sequences together form the genetic makeup of all species.

A protein is a chain of simple molecules called *amino acids* and is responsible for various functions and properties that characterize an organism. In nature, normally we find 20 different amino acids in a protein, but there are few cases when nonstandard amino acids are seen in a protein[30]. These proteins are assembled using the information encoded in associated DNA molecules. A DNA molecule is also composed of simpler molecules, called nucleotides, which are exactly 4 in number. Though the structure of DNA molecule is a twisted helix, the sequence of only one of the strands forming this helix is sufficient to describe the DNA (the other strand would be a *complement* strand, which can be determined from the original). Thus, both protein and DNA molecules are represented as strings of letters, with 20 alphabets for protein and 4 alphabets for DNA molecules.

The data model provides two classes, *DNAStr* and *ProteinStr*, to represent the sequence information of segment of a DNA molecule or a Protein molecule respectively.The storage representation of DNAStr and ProteinStr makes use of the small alphabet set, and encodes the DNA sequence using two bits and Protein sequence using 5 bits.

The behavioral part of this data model, provides functions for translation of DNA sequence into a Protein sequence[3] and vice-versa, complementary DNA strand generation and substring operations.

3.3.2 Similarity Search

Sequence comparison is the most important operation in computational biology, and it forms the basis for many other complex manipulations. The similarity searching over DNA and Protein sequences gives important clues about the genetic and phylogenetic relationships between species.

A fundamental concept in measuring similarity between two sequences is that of *alignment score* between two sequences or their subsequences. The alignment between two sequences is defined as the insertion of spaces in arbitrary locations along the sequences so that they end up with the same size, subject to the constraint that no space in one sequence be aligned with a space in the other. The scores for all possible combination over alphabets (including space) are provided as cost matrices, and the score of an alignment is

[3]Three nucleotides in DNA sequence determine the amino acid in the Protein sequence encoded by the DNA. And for every DNA sequence, there are three possible reading frames giving distinct Protein sequence, and all reading frames are simultaneously translated during this operation.

computed as the sum of scores of individual alphabet alignments in it.

There are many sequence similarity search algorithms based on various techniques [16, 30], but the most popular are *BLAST* [1] and *FastA* [23]. Both are heuristic based algorithms, based on the notion of *Local Alignment* [31], using the alignment scores of segments of sequences, as a measure of similarity between two sequences involved in the search.

Both these algorithms involve a complete scan of the sequence database for every query sequence. The index structures designed for nearest neighborhood searching in metric spaces cannot be used, as the similarity measures do not form a metric space since the scoring matrices contain negative costs [30]. Efficiency can be clearly improved by supporting these algorithms within the database rather than at the application layer.

BODHI provides these two algorithms, over DNA as well as protein sequences, and they can be part of an OQL query. The similarity is computed using a set of default cost matrices, (i) PAM-120 [8] matrices for proteins, and (ii) for DNA identities are scored with +5, mismatches as -4 following the scheme in [1].

4. IMPLEMENTATION DETAILS

As previously described in Section 3, the service modules of the BODHI, provide indexing methods, over object hierarchies and spatial data, and sequence similarity algorithms. In this section, we focus on the implementation details of these service modules with emphasis on index structures.

Shore supports extensions to the standard set of features it supports in the form of Value Added Servers(VAS). The storage manager provides a programming interface, called Storage Manager Programming Interface [6], for writing these extensions. The service modules in BODHI, are implemented at this layer.

4.1 Spatial and Sequence primitives

The modeling primitives for spatial and sequence data are introduced as declarations in SDL(SHORE Definition Language), data definition language provided by SHORE. The implementations of the operations on them are provided in separate C++ files, which are compiled and maintained separately as libraries. The schema manager and query processor of BODHI, have been programmed to consider these types as primitive datatypes.

4.2 Indexes over Objects

The index structures over class inheritance hierarchies and object relationship paths are part of the Object Services VAS. The interface with query processor consists of methods to create and destroy indexes, to retrieve and to scan the indexes. These methods are called by the query processor using RPCs which are evaluated by the VAS and the results are returned to query processor.

4.2.1 Path Dictionary

The implementation of PD-index for object relationship graph, extends the the s-expression based implementation suggested in [22] to handle the presence of N:M relationships between objects and compound aggregations such as bags, sets and sequences. The original implementation encodes all paths terminating at the same object into an *s-expression*. The s-expression for the path $C_1.C_2.C_3...C_n$ is defined as follows:

- $S_n = \theta_n$, where θ_n is the OID of an object in class C_n or null.
- $S_i = \theta_i(S_{i+1}[, S_{i+1}])$ $1 \leq i < n$, where θ_i is the OID of an object in class C_i or null, and S_{i+1} is an s-expression for the path $C_{i+1} \dots C_n$.

S_i is an s-expression at i^{th} level in which the list associated with θ_i contains recursively the OIDs of all descendant objects of θ_i. Except for the objects in C_n, every object on the path has a descendant list, which may be empty.

Due to the inherent tree structure, the s-expression scheme supports 1:1 and 1:N relationships naturally. But in models representing the bio-diversity data, the possibility of N:M relationships cannot be overlooked. To illustrate this, consider the relationship between the PlantSpecies and their associated GeoRegions, in the schema of Figure 2. Note that a PlantSpecies can be found in any number of GeoRegions and a single GeoRegion may harbor many PlantSpecies. This forms a natural N:M relationship between species and their geographical location.

Moreover, structures like *Bag* and *Sequence* are common in biological data models. Each of these relationships need different treatment at the implementation level to preserve the correctness and to reduce redundancy without compromising on retrieval time.

We have extended the implementation given in [22] to support these additional requirements. The main idea behind our extensions is to break the of N:M relationship into multiple 1:N relationships. But a straightforward application of this idea introduces complications in maintenance of s-expressions.

Supporting N:M relationships: Consider the representative N:M relationship graph shown in Figure 5(a). If we break this into multiple 1:N relationships, the graphs and the corresponding s-expressions look as in Figure 5(b). Note the redundancy in these s-expressions: The children of B_1 are replicated in both of the s-expressions of A_1 and A_2. This problem can be solved by using a flag in the entries of s-expression. The flag denotes whether the entry is a direct reference or an indirect reference. All the descendant entries of an OID will be stored only in the entry which contains direct reference to that OID. This modification is shown in form of a graph in Figure 5(c) with their corresponding s-expressions. Note that the suffix for each entry denotes whether it is a direct reference or an indirect reference. Though this modification, duplicates (with different flag values) the B_1 entry, we avoid duplicating the children of B_1, thus saving space.

Extensions to support Bags and Sequences: The above modification works fine for storing ordinary references and sets. But in presence of bags, further redundancy is possible. The example for this is shown in Figure 6. The number on edge from a to b denotes the number of times b appeared as a reference in the bag of a. The corresponding s-expressions for this

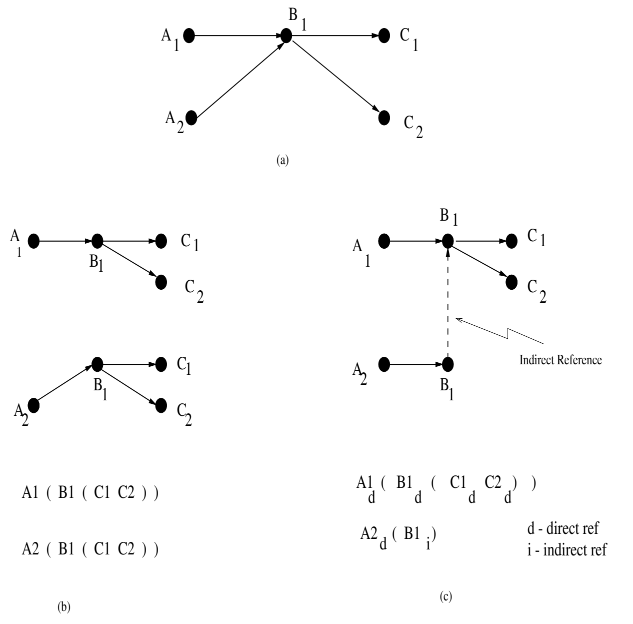

Figure 5: Representing N:M relationships (a) N:M relationship (b) Equivalent 1:N relationships with replicated paths (c) Equivalent 1:N relationships with indirect references.

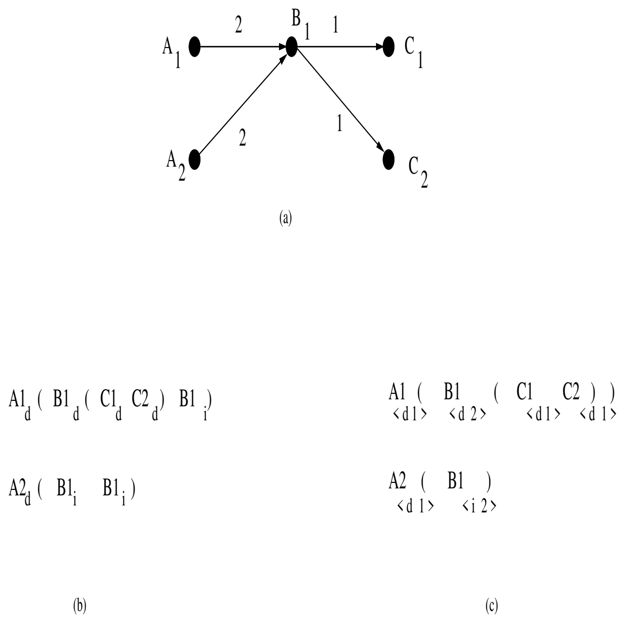

Figure 6: Representing N:M relationships in presence of (a) Bags (b) Equivalent 1:N relationships with indirect references (c) Equivalent 1:N relationships with indirect references as well as replication counts

graph using above implementation are given in Figure 6(b). Note that the entry of B_1 is repeated n times in each expression, where n denotes the number of times B_1 is referenced in parent object. This replication can be eliminated by introducing one more field in the entry of s-expression which stores this replication count. This reduces the storage overhead for storing bags since OIDs are not duplicated. The s-expressions with this modification are shown in Figure 6(c).

The implementation also supports sequences by maintaining the order of the children of a given parent in the s-expressions.

4.2.2 MT-Index

The MT-Index over type hierarchy, turns out to be a multi-dimensional index with type as a dimension, after the linearization of the inheritance hierarchy as proposed in [26]. SHORE storage manager already provides a multi-dimensional index, R^*-Tree. Hence, the MT-Index has been implemented using R^*-Trees with an additional dimension to represent the type of the object.

4.3 Spatial Services

Aiming to provide an efficient implementation of Spatial Services, we were faced with an operational problem. The Shore SM interface, the interface to extend the features provided by the storage manager, allows one to introduce new logical index structures over data but no page-level storage control is provided to index structures. This excludes the possibility of implmenting index structures, such as Hilbert R-Tree, that rely on physical clustering of data for performance reasons.

This forced us to implement the Hilbert R-tree by refactoring the existing code for R^*-Trees within Shore storage manager and exporting its interface to both VAS layer (through SM interface) and to Application layer (through SDL interface).

4.4 Sequence Services

As already discussed, the genome sequence support in BODHI, includes a pair of sequence similarity algorithms (BLAST and FastA) in addition to the data primitives to store and manipulate finished DNA and Protein sequences.

The Sequence services module provides an efficient storage representation of DNAStr and ProteinStr datatypes by encoding them using 2 bits and 5 bits per alphabet respectively (though further compression could be possible on these strings, it will significantly slowdown the similarity searching). Each sequence is stored in the form of a record of a file in the storage manager. This scheme of storage allows us to perform complete scans of the sequences in an efficient manner. In addition, we make use of the fine-granularity locking provided by the SHORE storage manager, to search even when a record is being updated and to perform multi-threaded searches.

The current implementation of BLAST and FastA algorithms, provide for simultaneous execution of two similarity searches over proteins and over DNA sequences.

5. QUERY PROCESSING

The query processor of BODHI integrates the features provided by the service modules through extended ODL/OQL for modeling and querying the database. In addition, it optimizes the queries using the metadata and index information, and as part of the *Client Interface Frame-*

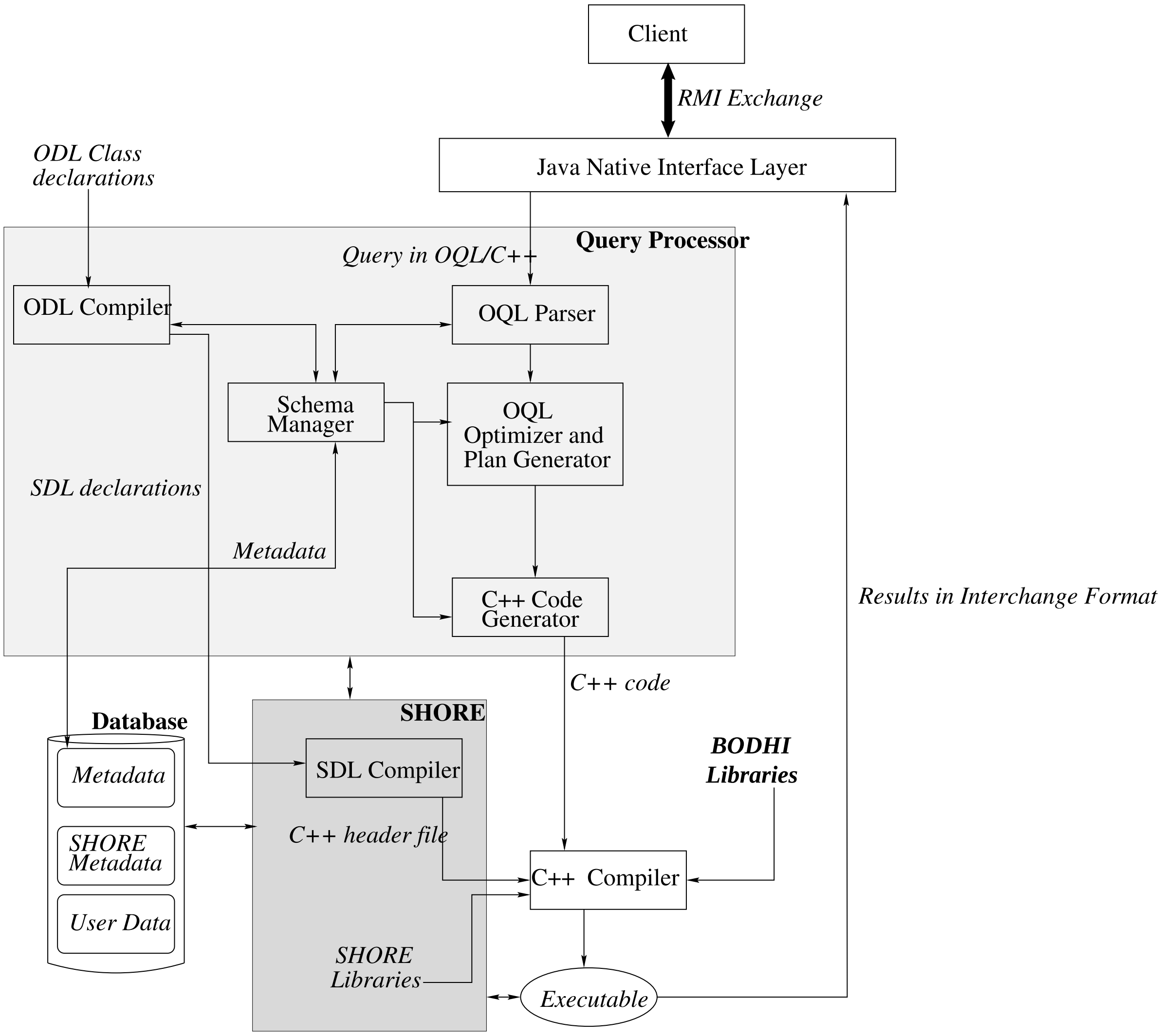

Figure 7: Schema Definition and Query Flow in BODHI

work(discussed in Section 6) it facilitates transformation of query results into an interchangeable format as requested by the user.

The core of the Query Processor module, is *lambda-DB*, a freely available, rule-based query processor and optimizer for ODL/OQL. Lambda-DB, performs all optimizations on the query at the compile-time, producing an corresponding executable, resulting in extremely fast query executions.

The schematic representing the flow of schema definitions and queries over the database, is illustrated in Figure 7.

5.1 Schema Definition

The schema is defined, as already mentioned, using ODL – with extensions necessary for BODHI. The schema declarations are first converted into SDL, before getting compiled into an C++ header file. During this phase, the *Schema Manager* of the query processor obtains the metadata needed for typechecking and optimization of queries and maintains it in the database. The implementation part of the schema declaration, is abstracted out into a C++ code and is available for compilation into a linkable library.

5.2 Query Flow

The BODHI consists of a web-enabled front end for querying with the query processor forming the back-end. The interface between two is provided through a *JNI*(Java Native Interface) layer, which receives the queries from clients through *RMI*(Remote Method Invocation).

The query strings obtained are type checked, parsed and compiled into C++ code with execution plans generated after incorporating the rules specified in the query optimizer. The type library of spatial and sequence data primitives and the implementations of various operations defined over them, which are precompiled into linkable libraries and header files, are linked to generate the executable of the query.

This executable contains implementations for interacting with the SHORE storage manager, Value Added Servers of Object and Sequence Services, and the implementation needed for transforming query results into interchangeable format. The transformed results are returned back, again through JNI layer, to the client.

5.2.1 Strategies for Query Optimization

The lambda-DB, provides a language, OPTL, which can be used to specify various optimization rules. The default ruleset that ships with lambda-DB, supports the standard optimizations including making use of B^+-Tree and R^*-Tree indexes in Shore. This ruleset has been enhanced to make use of specialized indexing schemes in BODHI.

Path Expression Computation As we already described in Section 3, due to the presence of object relationship paths, queries can be issued on any object deep in the relationship paths. So a query might involve computation of the predicate $C_1.a_1.a_2.....a_m = C_2.b_1.b_2......b_n$. This can be easily observed to be a join of the extents of the classes that form the path expression. So, one of the optimization schemes followed in lambda-DB, is to convert the path expression into a join[11], and then select from the resultant join table. But in BODHI, a specialized index structure, Path Dictionary, is provided for this purpose, which avoids the costly joins altogether.

A Path Dictionary can be built over most often queried paths in the application schema. But the identities of these paths is totally application dependent. The user can build a Path Dictionary using the extended OQL command for the purpose. The syntax of the command would look like *"create path index id1 on Species.statesFound.capital.location;"*.

Queries on Class Hierarchies The queries on class hierarchies, such those illustrated in Section 3.1, are not supported in the standard ruleset of the lambda-DB. BODHI provides Multi-key Type Index (MT-Index), for the purpose. Extra optimization rules are provided, which exploit the presence of an MT-Index over the class hierarchy. The user can optionally build an index on a class hierarchy rooted at object X, using the OQL command which would be like *"create mt index id2 on X"*.

Derived Attributes Derived attributes, represented as functions with no parameters, can be treated as though they are value attributes of the object and indexes can be maintained on them. Such functions can depend on other attribute values of the object, and such dependencies cannot be inferred from their implementation. In order to support indexing over derived attributes, we have extended the method declaration syntax of ODL, to specify a list of attributes on which a derived attribute depends.

Based on this information, whenever an object is created or the attributes of the object are modified, the query processor computes the function and updates the index maintained on it. The following shows a simple example illustrating how to define a function index. The class definition looks like:

```
class Forest (extent Forest_ext){
attribute string  name;
attribute polygon region;
float area() depends on region;
};
```

The OQL syntax to create the index over *area*, looks like:

```
create function index faind on Forest(area);
```

where *faind* is the index identifier.

6. CLIENT INTERFACE FRAMEWORK

As mentioned in Section 2, a desirable feature of a biodiversity data repository system is to support knowledge dissemination, visualization and the ability to exchange data with other similar databases over the Internet. The Client Interface Framework (CIF) component of BODHI provides for these needs.

The CIF consists of a software layer, called *Client Interface Server*(CI-server). that sits on top of the query processor. The CI-Server is responsible for transforming or formatting the query results. The CI-Server is separated from the query processor and can be easily extended to accommodate a new formatter, without having to change or even re-compile the query processor. In order to achieve this extendibility, we use a well known design pattern, *Strategy Pattern*[13]. A conceptual class diagram illustrating the usage of this design pattern is given in Figure 8. As shown in the diagram, the CI-Server provides a common interface called *Formatter Interface*, from which all other formatter objects are derived. When the Query Processor needs to emit the query results, it requests the CI-Server and obtains an object conforming to the Formatter interface i.e., an object of a formatter class. The implementation of this formatter object is provided in a dynamically loadable library. Since all implementations have a common interface, there is no need for changing the query processor for accommodating a new formatter class.

The implication of separating the data formatting functionality from the query processor is that, during data transformation, no metadata maintained by the Schema Manager is accessible to the CI-Server. But the data transformation needs the meta-data information associated with data. Therefore, a protocol of information exchange between the CI-Server and query processor is implemented for this purpose. This format of data interchange embeds both data and meta-data, and the implementation of Formatter objects takes care of formatting the results according to the transformation rules and returns it to the client. Further implementation details about the CIF are available in [21].

A useful byproduct of the CIF approach is that it does not distinguish between the two categories of "client"s, viz., the visualization client and databases on the Web that need to exchange data.

7. CURRENT STATUS

Having described the overall design and architecture of the BODHI system, we now overview the current status of our prototype implementation.

The major portion of BODHI is written in C++, the main exception being the query processor, which is implemented

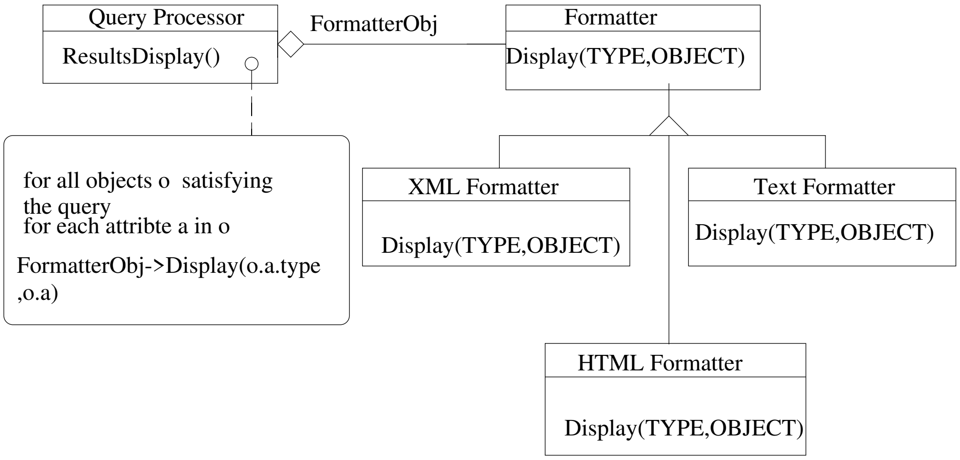

Figure 8: Query Processor and Result Formatters

in OPTL, the language provided by lambda-DB. Our initial implementation is on a Sun Ultra-1 Workstation running Solaris 2.5. The implementation has been going on for about a year and most of the key components of the system have been completed. The modules of spatial and sequence services have been tested, largely using synthetic data generated by us. Recently, in order to meet our goal of also supporting a lowcost system, we have ported the implementation of all the service modules onto an Intel Pentium-II machine, with GNU/Linux as the operating system. Current work includes, incorporating the genetic sequence similarity algorithms into the query language, implementing the spatial join optimizations and providing bulk loading facilities.

The client interface framework of BODHI was used to develop a browser based graphical visualization client. An XML/DTD, custom designed by combining the relevant portions of *Geography Markup Language* by OpenGIS[27] and *ANZMeta DTD Ver1.1*, was used for exchange of information with the client. The visualization client, implemented completely in Java, using the latest visualization features provided by Swing, has capabilities to display the aggregation graphs of the application, and to render the geographical maps using spatial co-ordinates sent by the database server.

We are currently working on populating the database with data already collected by the Center for Ecological Sciences at our institute and to conduct field trials on the useability and performance of the system.

8. RELATED WORK

Bio-diversity data consists of both macro-level and micro-level information ranging from ecological information to genetic makeup of organisms and plants. Apart from our work, we are not aware of any other that attempts to combine the complete spectrum of information, though the need for it is highlighted in a recent proposal for *GBIF*(Global Biodiversity Information Facility)[29] by OECD(Organization for Economic Co-operation and Development). This proposal identifies the domain level challenges in building a global, interconnected data repository of bio-diversity information system and notes that the urgent requirement in bio-diversity studies is a suitable information management architecture for handling vast amounts of diverse data.

In the area of macro-level bio-diversity data management, there have been many national level government efforts like *ERIN*[4], *INBio*[18] and global initiatives like *Species 2000*[32], *the Tree of Life*[25] etc. However, all these efforts have the following drawbacks in their information architectures : (1) Systems are based on the flat relational model, which does not seem to fit well with the bio-diversity data which is naturally hierarchical; (2) Little attention has been paid to supporting data exchange features (a recent exception is the XML/DTD proposal from ERIN, for ecological information exchange[2]); and (3) They do not support adhoc querying on their databases by independent clients.
In comparison, our work attempts to develop an architecture that addresses most of these deficiencies.

The micro-level bio-diversity data, or genetic information of various species, has been growing steadily due to a multitude of genome sequencing initiatives. The specific data management issues in handling such data[15, 14] have been addressed in quite a few proposals. In all of these proposals, the database management architecture has been tailored for the specific purposes of the project. For example, consider the *ACeDB* (A C.elegans Database)[9] database system, originally proposed for *C. elegans* genome sequencing project. ACeDB is an object oriented data management tool that has many features that make it a extremely popular software in many sequencing projects[33]. ACeDB handles missing data and schema evolution issues, common requirements in a ongoing sequencing projects, in a flexible manner. However, in spite of its popularity in genome sequencing community, it cannot be considered for the larger requirements of bio-diversity data handling due to the following reasons: (1) Its lack of support for geo-spatial data; (2) Weak support for database updates; and (3) The lack of recovery mechanisms necessary in large data repositories.

In BODHI, we have retained the key strengths of ACeDB (its sequencing algorithms and object-oriented basis), and augmented it with strong database functionalities, in addition to other features necessary for a complete bio-diversity information repository.

9. CONCLUSIONS

In this paper, we presented the design and implementation of BODHI, a data management system for plant biodiversity applications. This system, developed in close collaboration with domain experts of bio-diversity studies, is a state-of-the-art solution that provides a generic framework for the requirements of the domain at various levels of data management. Apart from efficiently integrating many datatypes, from genetic to ecological details, BODHI lends itself for flexible data interchange between repositories over the internet. The scalability and the low cost (free!) of the system make it suitable for both large and small data collection efforts. The bio-diversity database of flora of Western Ghats is currently being built over BODHI, intended for use at the *Center for Ecological Studies, Indian Institute of Science*.

We are currently working on improving the efficiency of the genome sequence similarity and pattern matching operations on large volumes of genetic data. In addition, we plan to use BODHI as a platform for further research in various pattern discovery processes over spatial and genome sequence data.

10. ACKNOWLEDGEMENTS

This work was supported by a research grant from the Dept. of Bio-technology, Govt. of India, for which Prof. V. Nanjundiah and Prof. Madhav Gadgil of the Centre for Ecological Sciences are the co-investigators. The following graduate students from the Dept. of Computer Science & Automation participated in the system design and implementation: Madhava Rao Cheethirala, Amitabha Ghosh, Satheesh Konidala, Rajarao Tadimeti. We thank Prof. Ramesh Hariharan, also of the Dept. of Computer Science & Automation, for his support and encouragement.

11. REFERENCES

[1] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. A basic local alignment search tool. *Journal of Molecular Biology*, (215), 1990.

[2] Anzmeta DTD version 1.1. http://www.erin.gov.au/database/metadata/anzmeta/anzmeta-1.1html.

[3] Biopolymer markup language - BIOML. http://www.bioml.com/.

[4] T. Boston and D. Stockwell. Interactive species distribution reporting, mapping and modeling using the world wide web. In *Proc. of the 2nd Intl. WWW Conf.*, October 1994.

[5] T. Brinkhoff, H. P. Kriegel, and B. Seeger. Efficient processing of spatial joins using R-trees. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, May 1993.

[6] M. J. Carey et al. Shoring up persistent applications. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, May 1994.

[7] R. Cattel, editor. *The Object Database Standard: ODMG-93*. Morgan-Kaufmann Publishers, 1994.

[8] M. Dayhoff, R. Schwartz, and B. Orcutt. A model of evolutionary change in proteins. *Atlas of Protein Sequence and Structure*, 5, 1978.

[9] R. Durbin and J. Thierry-Mieg. A C.elegans database documentation. ftp://ncbi.nlm.nih.gov/.

[10] C. Faloutsos and S. Roseman. Fractals for secondary key retrieval. In *Proc. of the ACM SIGACT/ SIGMOD Symposium on Principles of Database Systems*, March 1989.

[11] L. Fegaras. An experimental optimizer for oql. Technical Report TR-CSE-97-007, University of Texas at Arlington, 1997.

[12] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2), 1998.

[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patters: Elements of Reusable Object-oriented Software*. Addison-Wesley Publishing Company, 1995.

[14] N. Goodman, S. Rozen, and L. Stein. A glimpse at the DBMS challenges posed by the human genome project. ftp://genome.wi.mit.edu/pub/papers/Y1994/challenges.ps.Z.

[15] N. Goodman, S. Rozen, and L. Stein. Building a laboratory information system around a c++-based object oriented dbms. In *Proc. of the 20th Intl. Conf. on Very Large Databases*, September 1994.

[16] D. Gusfield. *Algorithms on Strings, Trees and Sequences : Computer Science and Computational Biology*. Cambridge University Press, 1997.

[17] R. Güting. An introduction to spatial database systems. *VLDB Journal*, 3(4), 1994.

[18] Inbioparque. http://www.inbio.ac.cr/en/default.html.

[19] H. Jagadish. Linear clustering of objects with multiple attributes. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, May 1990.

[20] I. Kamel and C. Faloutsos. Hilbert r-tree: An improved r-tree using fractals. In *Proc. of the 20th Intl. Conf. on Very Large Databases*, September 1994.

[21] S. Konidala. Query processing and optimization in oshadhi. Master's thesis, Department of Computer Science & Automation, Indian Institute of Science, Jan. 2000.

[22] W. Lee and D. Lee. Path dictionary: A new access method for query processing in object-oriented databases. *IEEE Transactions on Knowledge and Data Engineering*, 10(3), May 1998.

[23] D. Lipman and W. Pearson. Rapid and sensitive protein similarity searches. *Science*, (227), 1985.

[24] M. Lo and C. Ravishankar. Spatial joins using seeded trees. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, May 1994.

[25] D. Maddison and W. P. Maddison. The tree of life: A multi-authored, distributed internet project containing information about phylogeny and biodiversity. http://phylogeny.arizona.edu/tree/phylogeny.html, 1998.

[26] T. Mück and M. Polaschek. A configurable type hierarchy index for oodb. *VLDB Journal*, 6(4), 1997.

[27] Geography markup language (gml) v1.0. http://www.opengis.org/techno/specs/00-029.pdf.

[28] R. Pankhurst. *Practical Taxonomic Computing*.

Cambridge University Press, 1991.

[29] H. Saarenmaa. The global biodiversity information facility: Architectural and implementation issues. Technical Report TR-34, European Environment Agency, 1999.

[30] J. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology.* PWS Publishing Company, 1997.

[31] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, (147), 1981.

[32] Species2000. http://www.species2000.org/.

[33] L. Stein and J. Thierry-Mieg. ACeDB: A genome database management system. *Computing in Science and Engineering*, 1(3), 1999.

[34] R. Vidya and J. Haritsa. Oshadhi : A biodiversity information system. In *Proc. of the 7th International Conference on Management of Data*, 1995.