Research

Accessing and distributing EMBL data using CORBA (common object request broker architecture)

Lichun Wang, Patricia Rodriguez-Tomé, Nicole Redaschi, Phil McNeil, Alan Robinson and Philip Lijnzaad

Address: EMBL Outstation - Hinxton, European Bioinformatics Institute, Wellcome Trust Genome Campus, Hinxton, Cambridge CB10 1SD, UK.

Correspondence: Lichun Wang. E-mail: lcwang@ebi.ac.uk

Published: 6 November 2000

Genome ***Biology*** 2000, **1(5)**:research0010.1–0010.10

The electronic version of this article is the complete one and can be found online at http://genomebiology.com/2000/1/5/research/0010

Received: 26 July 2000
Revised: 6 September 2000
Accepted: 21 September 2000

Abstract

Background: The EMBL Nucleotide Sequence Database is a comprehensive database of DNA and RNA sequences and related information traditionally made available in flat-file format. Queries through tools such as SRS (Sequence Retrieval System) also return data in flat-file format. Flat files have a number of shortcomings, however, and the resources therefore currently lack a flexible environment to meet individual researchers' needs. The Object Management Group's common object request broker architecture (CORBA) is an industry standard that provides platform-independent programming interfaces and models for portable distributed object-oriented computing applications. Its independence from programming languages, computing platforms and network protocols makes it attractive for developing new applications for querying and distributing biological data.

Results: A CORBA infrastructure developed by EMBL-EBI provides an efficient means of accessing and distributing EMBL data. The EMBL object model is defined such that it provides a basis for specifying interfaces in interface definition language (IDL) and thus for developing the CORBA servers. The mapping from the object model to the relational schema in the underlying Oracle database uses the facilities provided by Persistence™, an object/relational tool. The techniques of developing loaders and 'live object caching' with persistent objects achieve a smart live object cache where objects are created on demand. The objects are managed by an evictor pattern mechanism.

Conclusions: The CORBA interfaces to the EMBL database address some of the problems of traditional flat-file formats and provide an efficient means for accessing and distributing EMBL data. CORBA also provides a flexible environment for users to develop their applications by building clients to our CORBA servers, which can be integrated into existing systems.

Background

The EMBL (European Molecular Biology Laboratory) Nucleotide Sequence Database (often referred to as the EMBL database) [1] is hosted at the European Bioinformatics Institute (EBI). It is a comprehensive database of DNA and RNA sequences that are directly submitted from researchers and genome sequencing groups, and collected from the scientific literature and patent applications. It is produced in an international collaboration with GenBank (NCBI, Bethesda, USA) and DDBJ (the DNA Data Bank of

Japan, CIB, Mishima, Japan). Each of the three collaborating groups collects a portion of the total sequence data reported worldwide, and all new and updated database entries are exchanged daily. The amount of sequence data is growing exponentially.

As our scientific understanding deepens, the complexity of the related information increases as well. As a result, the structure of the data also keeps changing. The EMBL database is managed and maintained using the relational database management system (DBMS) Oracle. It contains over 130 tables and 140 relationships, having around 80 Gigabytes (Gb) of data comprising nearly 10 million objects of primary data and millions of sub-objects called 'features'. Traditionally, the sequences and related information, which have been collected over a long period of time, are made available in flat-file format via ftp, CD-ROM, www tools, and so on. The queries through tools such as SRS (Sequence Retrieval System, a network browser for databanks in molecular biology) [2] also return data in flat-file format. However, flat files have a number of shortcomings: the format may not be described formally; it is difficult to represent complex data and relationships, the meaningful units of information ('objects') are not represented or handled well; it is hard to retrieve objects separately; assembly of objects into bigger aggregates is difficult; elaborate parsing is often required; and so on. In general, the current availability of the resources is not matched by a flexible environment to meet individual researchers' needs.

An industry standard, the Object Management Group's (OMG) common object request broker architecture (CORBA), provides platform-independent programming interfaces and models for portable distributed object-oriented computing applications [3-6]. Its independence from programming languages, computing platforms and network protocols provides a solution for developing new applications for querying and distributing biological data [7-13], which can also be integrated into existing systems. Here we present a CORBA infrastructure developed at EMBL-EBI and show that the CORBA interfaces to the EMBL database address some of the limitations of the flat-file format and provide an efficient means for accessing and distributing EMBL data. CORBA also provides a flexible environment for users to develop application programs (for example, for sequence analysis or data mining).

Results and discussion

EMBL object data model

The diversity and structure of biological data complicate their use. To develop a CORBA server that provides access to our biological data, we need a well-defined object model to model the real-world biological entities, that is, to describe the structure and constraints present in the data, as well as how the data can be accessed and queried. It is a specification of the data in the problem domain, independent of how the actual database is implemented. This model can then be expressed in IDL (interface definition language) interfaces to the CORBA server at one end, and mapped to a database schema for the underlying data management and storage at the other end.

We use the unified modeling language (UML) notation [14,15] for this model. According to the UML, a model is organized into packages. A package groups classes that are semantically related and a dependency indicates that a package uses classes from another package. Each class belongs to exactly one package. A class is a descriptor for a set of objects that have similar properties, behavior and relationships to other objects. An attribute is a named property of a class. An attribute can be derived, that is, its value is computed from the values of other attributes. An operation is a procedure attached to a class, describing the behavior of the class. A class can be created by inheriting all attributes and operations from one or more of its superclasses. An association is a description of links or a set of links that specify connections among objects. An association can be reflexive, connecting a class with itself. Multiplicity defines how many times one object may link to another through an association. An association class has both association and class properties. It can be seen as an association that also has class properties, or as a class that also has association properties. It holds data that are relevant for the association, but for neither of the associated classes alone. A structured (composite) data type is represented as a class (as usual in object-oriented (OO) modeling). A class is used as an attribute type mainly if its role is solely to bundle simple data into a composite type (for example, Date). Multi-valued attributes are represented by instances of a parameterized class Coll{Type} (for example, Coll{string}).

Our object model of the EMBL database is organised into five main packages, as shown in Figure 1, where each package holds a set of closely related classes with a common purpose. The packages are: Sequence Info, classes representing biological sequences, general information about these sequences and administrative data associated with database entries; Feature Info, classes representing detailed sequence annotation (known as sequence features); Reference Info, classes representing bibliographic references that hold information about the sequences; Taxonomy Info, classes representing the taxonomy of the organisms from which the sequences were obtained; Location Info, classes representing locations on sequences.

There is one additional package, Types, which holds classes representing all the special data types used in various parts of the model. Each package contains a relatively isolated part of the entire object model, and is a clear candidate for re-use in models for other databases.

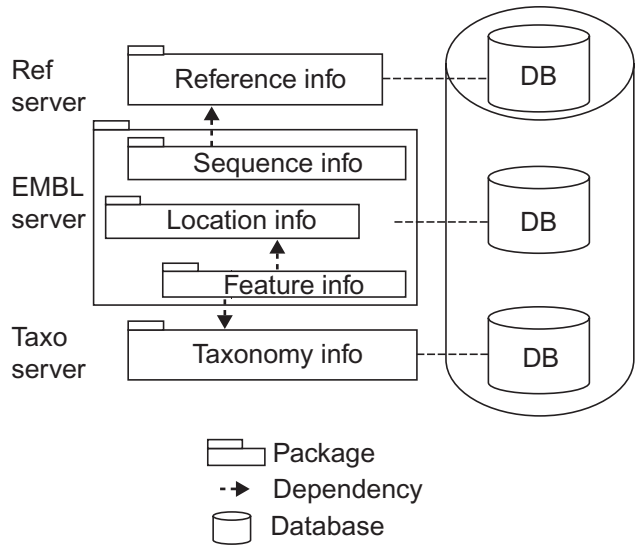

Figure 1
The database partitioning. The database is divided into five main packages: *Sequence Info*, all general information about sequences; *Feature Info*, detailed sequence annotation; *Reference Info*, bibliographic references; *Taxonomy Info*, the taxonomy of the organisms from which the sequences were obtained; *Location Info*, representing locations on sequences.

Figure 2 gives the definition of Sequence Info only. A full definition of the EMBL Nucleotide Sequence Database object model [16] can be found in Additional data file 1 with the online version of this article. The package Sequence Info defines class BioSeq, which represents biological sequences, and class SeqInfo, which describes general information about these sequences. The administrative data associated with database entries are defined in EntryInfo. The biological classes of sequence NSDBSeq, for nucleotide sequences, and PIDSeq, for protein sequences, are subclasses of BioSeq. VirtualSeq and PhysicalSeq are storage classes of sequence, that is, virtual or literal.

The definition of some biological entities is prone to change because of the rapid developments in molecular biology. Any change made to the structure of the model needs to be propagated to both the IDL specification that defines the CORBA server interfaces and underlying relational schema. To handle this problem, a strategy of using both explicit model and meta model is employed in defining Feature Info. The structure of the model is therefore not affected by changes to the feature definition, which makes it suitable for defining stable IDL interfaces.

The EMBL CORBA server mainly covers Sequence Info, Location Info and Feature Info, which are grouped into a big package that also includes Reference Info and Taxonomy Info. The reference and taxonomy servers are independent servers for Reference Info and Taxonomy Info. This paper is focused on the EMBL server.

System architecture and CORBA development

The system architecture is shown in Figure 3. On the server side, CORBA implementation objects access and query the relational database via PersistenceTM (Persistence Software) [17], which acts as a middleware between our CORBA implementation and the Oracle database. To allow invoking operations on the objects, the server provides its clients interfaces in OMG IDL, which is independent of the server implementation. An object's interface is composed of the operations and types of data that can be passed to and from those operations. Clients access the CORBA objects via operation calls through an object request broker (ORB), where the distribution details are handled by the ORB.

The CORBA development overview is shown in Figure 4. CORBA object interfaces together with their operations and type of data are defined in IDL. For the ORB, we have chosen IONA Inc's C++ ORB, Orbix™ [18]. Its IDL compiler generates skeleton code and stub code in C++. We provide the server object implementation code and the Persistence application code. These codes are subsequently compiled and linked together to become executable. Clients can be written in any language for which an ORB and IDL compiler are available, including Ada, C, C++, COBOL, CommonLisp, Eiffel, Java, Python, Perl, SmallTalk, Tcl, and so on. Note that we do not use the new features in CORBA 2.3, as ORBs that implement CORBA 2.3 have only become available recently.

IDL definition

The OMG IDL is CORBA's fundamental abstraction mechanism for separating object interfaces from their implementations [3,4]. It allows object interfaces to be defined in a manner that is independent of any particular programming language. It establishes a contract between client and server that describes the types and object interfaces used by an application. IDL definitions focus on object interfaces, the operations supported by those interfaces, and exceptions that might be raised by the operations. As data can only be exchanged between client and server if their types are defined in IDL, typically a large part of an IDL is concerned with the definition of data types. An interface can inherit from one or more other interfaces.

Following the EMBL data model, the IDL definition for the EMBL server comprises three IDL files: nsdb.idl, seqdb.idl and types.idl. The nsdb.idl defines the EMBL-specific sequences and related information and includes seven interfaces in the module nsdb: EntryInfo, Embl, EmblSeq, NucSeq, NucFeature, Location and FeatureLocation. The seqdb.idl defines the module seqdb that includes three interfaces: BioSeq, SeqInfo and Feature, which contain more general biological sequence information. The nsdb.idl and seqdb.idl use basic types defined in types.idl.

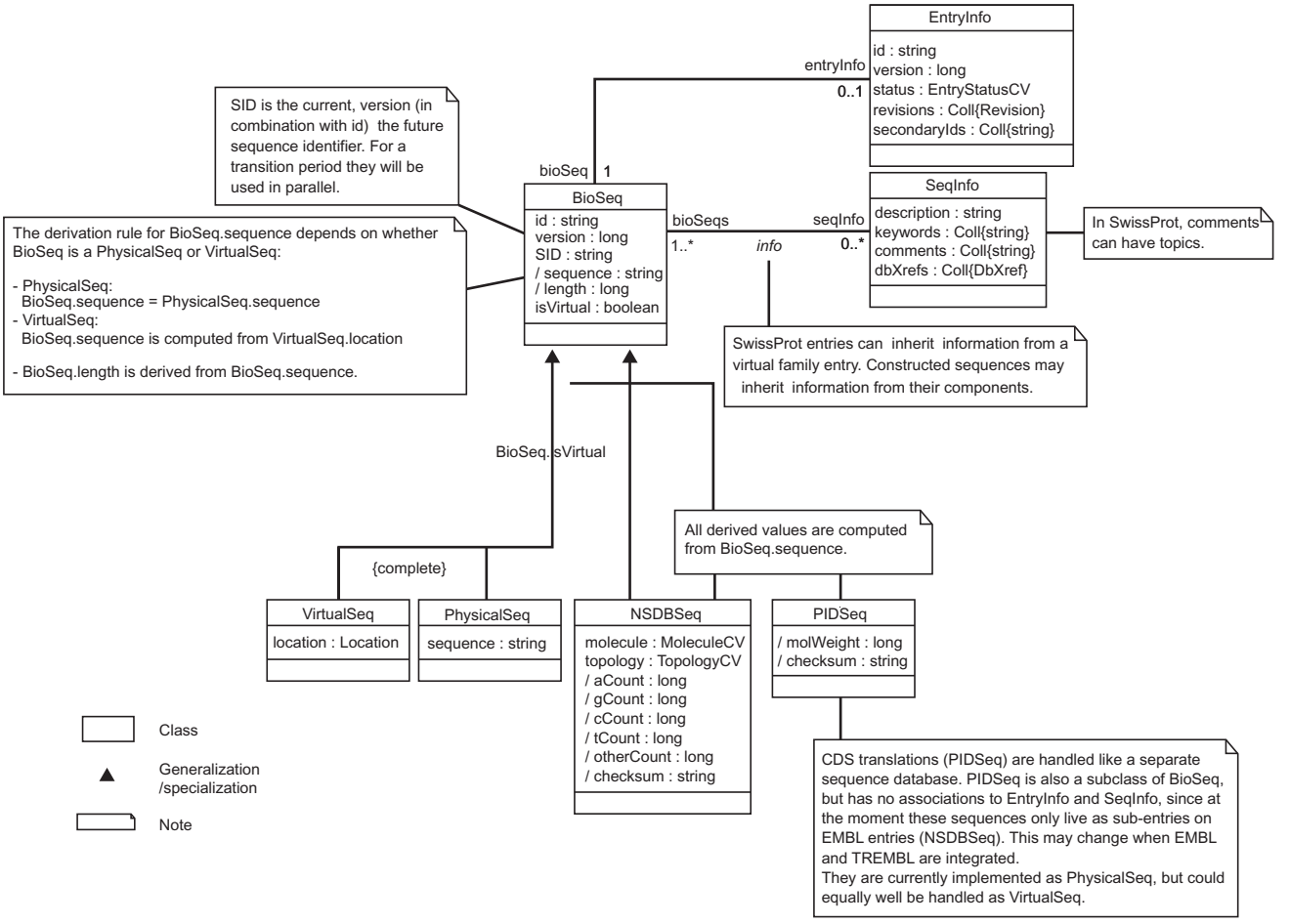

Figure 2
Sequence Info. This package defines class *BioSeq*, which represents biological sequences, and class *SeqInfo*, which describes general information about these sequences. The administrative data associated with database entries are defined in *EntryInfo*. The biological classes of sequence *NSDBSeq*, which is for nucleotide sequences, and *PIDSeq,* which is for protein sequences, are subclasses of *BioSeq*. *VirtualSeq* and *PhysicalSeq* are storage classes of sequence, that is, virtual or literal.

To reflect the accessing and querying of data, operations are defined in such a way that the return values of the operations represent attributes in the EMBL object data model. This supports 'creating objects on demand'. These objects are instances of 'data classes', which are the results of queries. Figure 5 gives the IDL specification of interfaces BioSeq and SeqInfo in module seqdb, and EntryInfo, NucSeq and EmblSeq in module nsdb (the full IDL definition can be found in the Additional data file with the oonline version of this article) [19]). NucSeq inherits from seqdb::BioSeq and EmblSeq inherits from NucSeq, seqdb::SeqInfo, and EntryInfo.

Class relationships

Each IDL interface is mapped into a class in C++ by the ORB's IDL compiler (in our case, the ORB is IONA's Orbix), and operations are mapped to member functions of the class. For the above interfaces BioSeq and SeqInfo, we have two mapped classes as shown in Figure 6.

The module itself is also mapped into a class here. Although it can normally be mapped to a namespace, our C++ compiler (Sun's SparcWorks 4.2) does not support namespaces. The relationship between classes is shown in Figure 7 in UML notation. The seqdb consists of three classes: SeqInfo, BioSeq, and Feature. The nsdb class comprises four classes: EmblSeq, NucFeature, FeatureLocation and Embl. EmblSeq inherits from the classes of EntryInfo, SeqInfo and NucSeq that in turn inherits from BioSeq. FeatureLocation inherits from Location, and NucFeature inherits from Feature.

Object-relational mapping

The EMBL CORBA server provides its clients with an object-oriented interface to the EMBL database. To achieve this, the

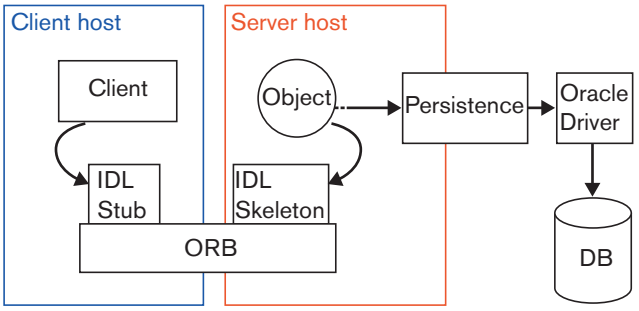

Figure 3
System architecture. On the server side, CORBA implementation objects access and query the relational database via Persistence™, which is a middleware between our CORBA implementation and the Oracle database. To allow invoking operations on the objects, the server provides its clients interfaces in OMG IDL, which is independent of the server implementation. An object's interface is composed of the operations and types of data that can be passed to and from those operations. Clients access the CORBA objects via operation calls through an Object Request Broker (ORB) where the distribution details are handled by the ORB.

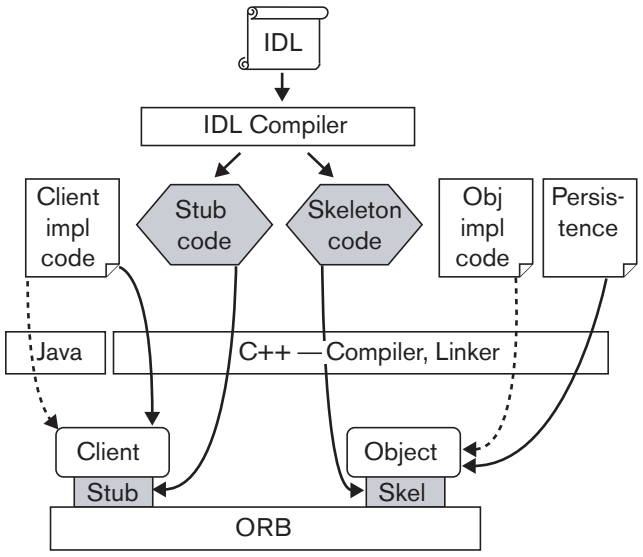

Figure 4
CORBA development overview. CORBA object interfaces together with their operations and type of data are defined in IDL. For the ORB, we have chosen IONA Inc's C++ ORB, Orbix™. Its IDL compiler generates skeleton code and stub code in C++. We provide the server object implementation code and the Persistence application code. These codes are subsequently compiled and linked together to become executable. Clients can be written in any language for which an ORB and IDL compiler are available, including Ada, C, C++, COBOL, CommonLisp, Eiffel, Java, Python, Perl, SmallTalk, Tcl, and so on.

object model needs to be mapped to the schema of the underlying Oracle relational database.

Persistence™, an object/relational tool from Persistence Software [17], is a mediator for transforming object operations to relational database calls and vice versa. It maps objects to relational rows and manages the objects in a shared cache, called the live object cache. It uses a proprietary object model description that maps classes to tables, objects to rows, attributes to columns and associations to foreign keys.

For inheritance relationships (single inheritance only), Persistence insists on a so-called horizontal mapping for performance reasons; that is, in the class hierarchy, only leaf nodes are represented by real database tables (or views). Non-leaf class objects are obtained as projections of the leaf class tables. However, for maximum flexibility, our existing database schema, which is independent and developed prior to the CORBA development, uses the so-called vertical mapping. In this case, each node in the class hierarchy has its own table, with subclass tables having no superclass attributes; their primary keys are also foreign keys to the superclass table. As our objects provide read-only access, it is possible to set up relational views that transform our tables into the horizontal object-to-relational mapping that is required by Persistence. This allows developers to create hierarchies of related objects from 'flat' tables. For example, for the class NsdbSeq that is inherited from BioSeq, its so-called Persistence horizontal object-to-relational mapping (view) is shown in Figure 8. This view is built from a number of tables (or views) as shown in Figure 9. The CORBA class EmblSeq is mapped to the view of NsdbSeq.

Using the data model and schema description, Persistence can also offer an automatic generation of an IDL specification as well as a complete CORBA server. We have not used this facility, however, as we want full control over the IDL specification. This approach has an advantage in using views even when a one-to-one mapping from IDL to EMBL tables remains. When the structure of data changes at the database side as a result of the increasing complexity of biological data or the availability of new modeling capability in the database, we need only a change on the underlying views. We much less frequently require a change on the Persistence mapping and code as these can still map the changed tables or data to the same objects at the CORBA side.

Object management

A 'live object cache' is a notion used in Persistence [17,20]. The basic model for managing live objects is to cache data instances read from the database, to register their primary key values, and to respond to queries based on the cached

```
module seqdb {

  interface BioSeq {
  string getBioSeqId();
  unsigned long getLength();
  any getAnySeq();
  unsigned long getBioSeqVersion();
  };

  interface SeqInfo {
  string getDescription() raises (type::NoResult);
  type::stringList getKeywords() raises (type::NoResult);
  type::stringList getComments() raises (type::NoResult);
  type::DbXrefList getDbXrefs() raises (type::NoResult);
  type::DbXrefList getReferences() raises (type::NoResult);
  };

  };

module nsdb {

  interface EntryInfo {
  string getEntryName();
  unsigned long getEntryVersion();
  string getEntryStatus();
  type::RevisionList getRevisions();
  type::stringList getSecondaryIds();
  unsigned long getCountA();
  unsigned long getCountC();
  unsigned long getCountG();
  unsigned long getCountT();
  unsigned long getOtherCount();
  }

  interface NucSeq : seqdb::BioSeq {
  string getSeq();
  unsigned long getCheckSum();
  string getTopology();
  string getMoleculeType();
  NucFeatureList getNucFeatures() raises (type::NoResult);
  Location getLocalLocation(in NucFeature nuc_feature) raises (type::InvalidRelation);
  type::DbXrefList getOrganisms() raises (type::NoResult);
  NucFeatureList getNucFeaturesByKey(in string key) raises (type::NoResult, type::InvalidArgumentValue);
  string getSubSeq(in unsigned long start, in unsigned long end) raises (type::IndexOutOfRange);
  string getSubSeqByFeature(in NucFeature feature) raises (type::InvalidRelation,   InexactLocation);
  }

  interface EmblSeq : NucSeq, seqdb::SeqInfo , EntryInfo {
  };
  ...
 }
```

Figure 5
Part of module *seqdb* and *nsdb*, extracted from the EMBL IDL specification. The interfaces *BioSeq* and *SeqInfo* are defined in module *seqdb;* The interfaces *EntryInfo, NucSeq* and *EmblSeq* are defined in module *nsdb. NucSeq* inherits from *seqdb::BioSeq* and *EmblSeq* inherits from *NucSeq, seqdb::SeqInfo,* and *EntryInfo.*

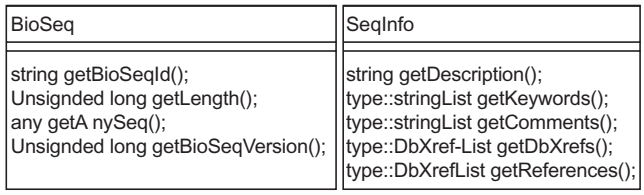

Figure 6
Classes of *BioSeq* and *SeqInfo*. The class BioSeq has four methods: getBioSeqId, getLength, getAnySeq and getBioSeqVersion. The returned values of methods represent attributes of class BioSeq defined in the object model, providing information on biological sequences. The class SeqInfo has methods: getDescription, getKeywords, getComments, getDbXrefs and getReferences, representing general information on the sequences.

data. As tuples are retrieved from the database, they are converted to objects and 'knitted' together according to the object-model mapping to form a network of in-memory objects. A live object cache maps information from relational tables into objects. Accessing and manipulating these objects in the live object cache is faster than querying the relational database, speeding up application performance considerably. Persistence can also ensure data integrity with appropriate locking and transaction management.

There are roughly three types of objects involved here: persistent objects, live objects, and CORBA objects. Here a persistent object is referred as a 'data object' in the database. A live object is an in-memory object in the live object cache. A CORBA object is a CORBA implementation object defined in IDL. Creation of a CORBA object is called instantiation. When a persistent object is loaded into memory, it becomes a live object. A CORBA object owns one or more live objects. Note that the ORB's object adapter, no matter whether a basic object adapter (BOA) or portable object adapter (POA), only serves as the glue between CORBA objects and the ORB. It is an object that adapts the interface of one object to a different interface expected by a caller and allows the caller to invoke requests on an object without knowing the object's true interface. Although the future CORBA may include garbage collection, the management of objects is currently at the application developer's discretion. This section discusses the management of objects.

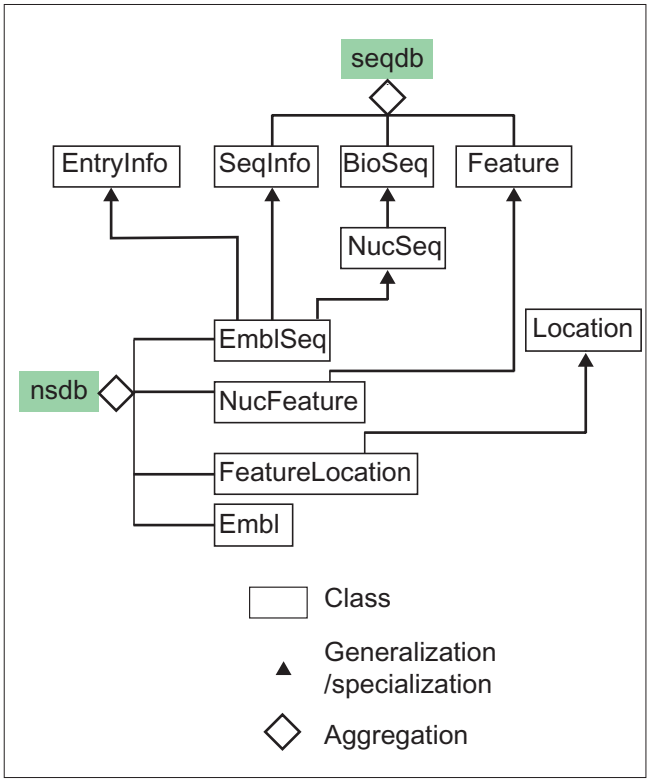

Figure 7
Class relationship in UML notation. The class *seqdb* consists of 3 classes: *SeqInfo*, *BioSeq*, and *Feature*. The class *nsdb* comprises 4 classes: *EmblSeq, NucFeature, FeatureLocation* and *Embl. EmblSeq* inherits from the classes of *EntryInfo, SeqInfo* and *NucSeq* that in turn inherits from *BioSeq. FeatureLocation* inherits from *Location*, and *NucFeature* inherits from *Feature*.

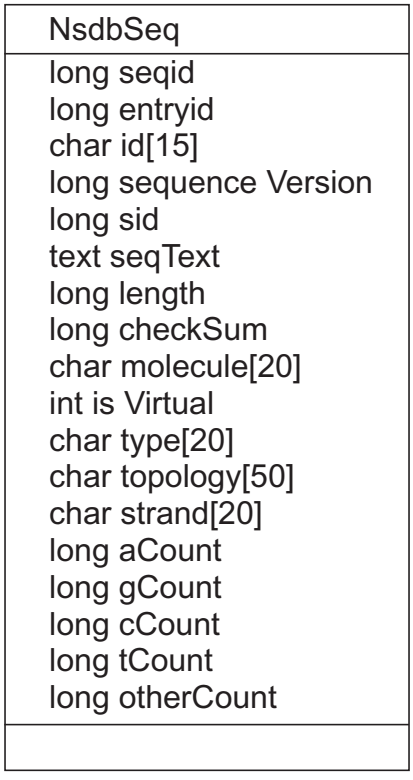

Figure 8
The NsdbSeq view in Persistence. As the class NsdnSeq inherits from the class BioSeq, the view of NsdbSeq in Persistence therefore has attributes defined in both NsdbSeq and BioSeq, representing information on nucleotide sequences.

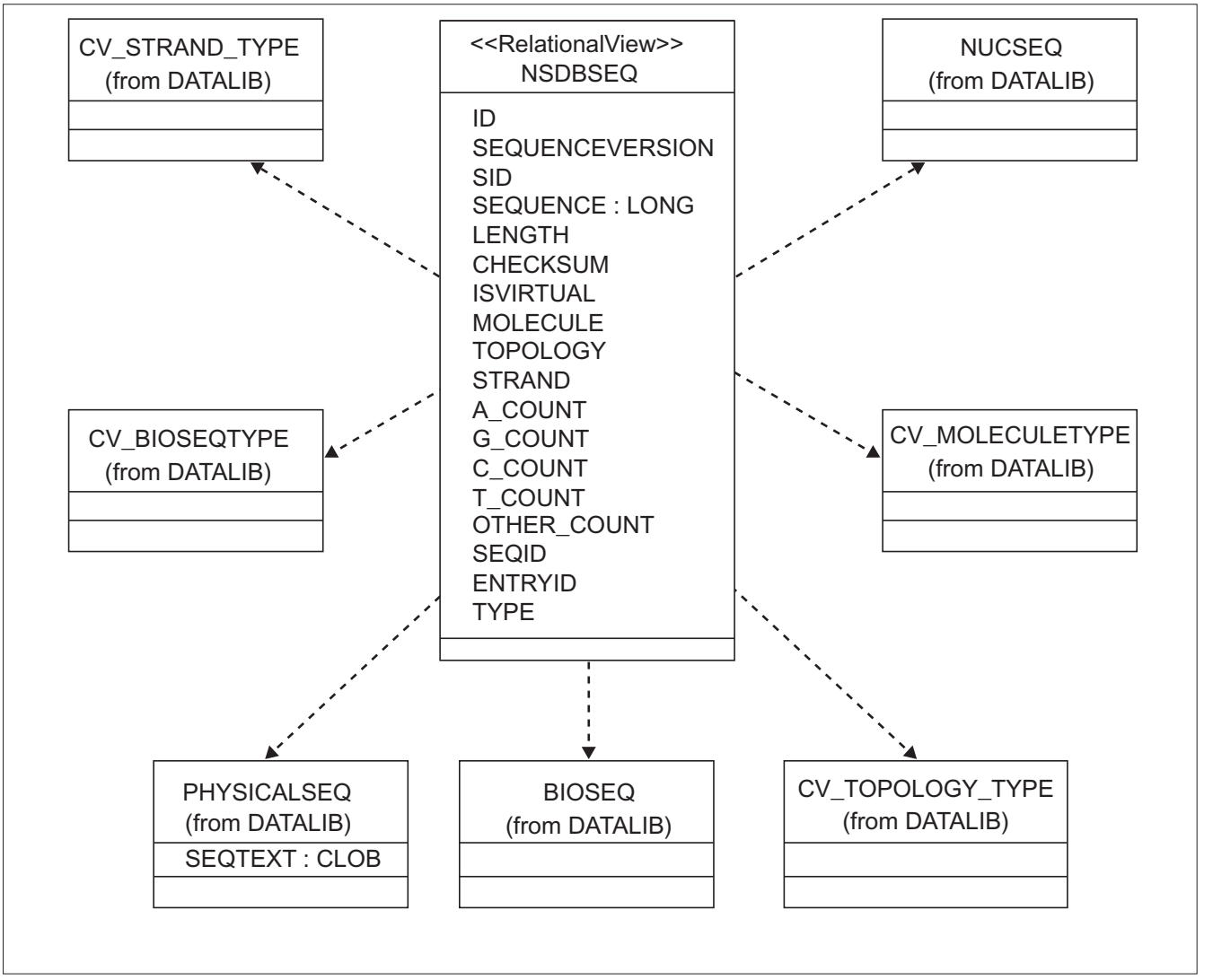

Figure 9
Tables for the NsdbSeq view in the actual database. The view is built from a number of tables (or views) of the database.

Creation of live objects

Data kept in the database are only loaded into the cache on an as-needed basis. We employ Orbix's loader techniques [18,21] together with Persistence's live object caching [17,20] to build our loaders to support the creation of objects in the live object cache. When an operation invocation arrives at the process, Orbix ORB searches for the target object in the process's object table. Loaders are called when an object reference enters an address space via a function findMe, and Persistence live objects are then loaded. If no live objects available in the live object cache respond to the call, Persistence will create a new live object via querying and accessing the relational database.

Eviction of CORBA objects

Objects are created at the clients' request. When the server has been running for some period of time, possibly weeks or months, it will have created a number of CORBA objects, which in turn contain a number of Persistence live objects, and will consume the memory space. Some of them will not be needed any more. The evictor pattern [3] describes a general strategy for limiting memory consumption. The basic idea is that we use an object manager to instantiate objects on demand. However, instead of blindly instantiating a new object every time, the object manager checks instantiated objects in the pool that it manages. If the called object is already in the pool, it can be used directly. If not, it will check if the number of objects has reached a specified limit. If so, the object manager will evict an older instantiated object and then instantiate a new one for the current request. Consequently, the older related Persistence live objects will also be deleted from the cache and new live objects will be loaded in if requested.

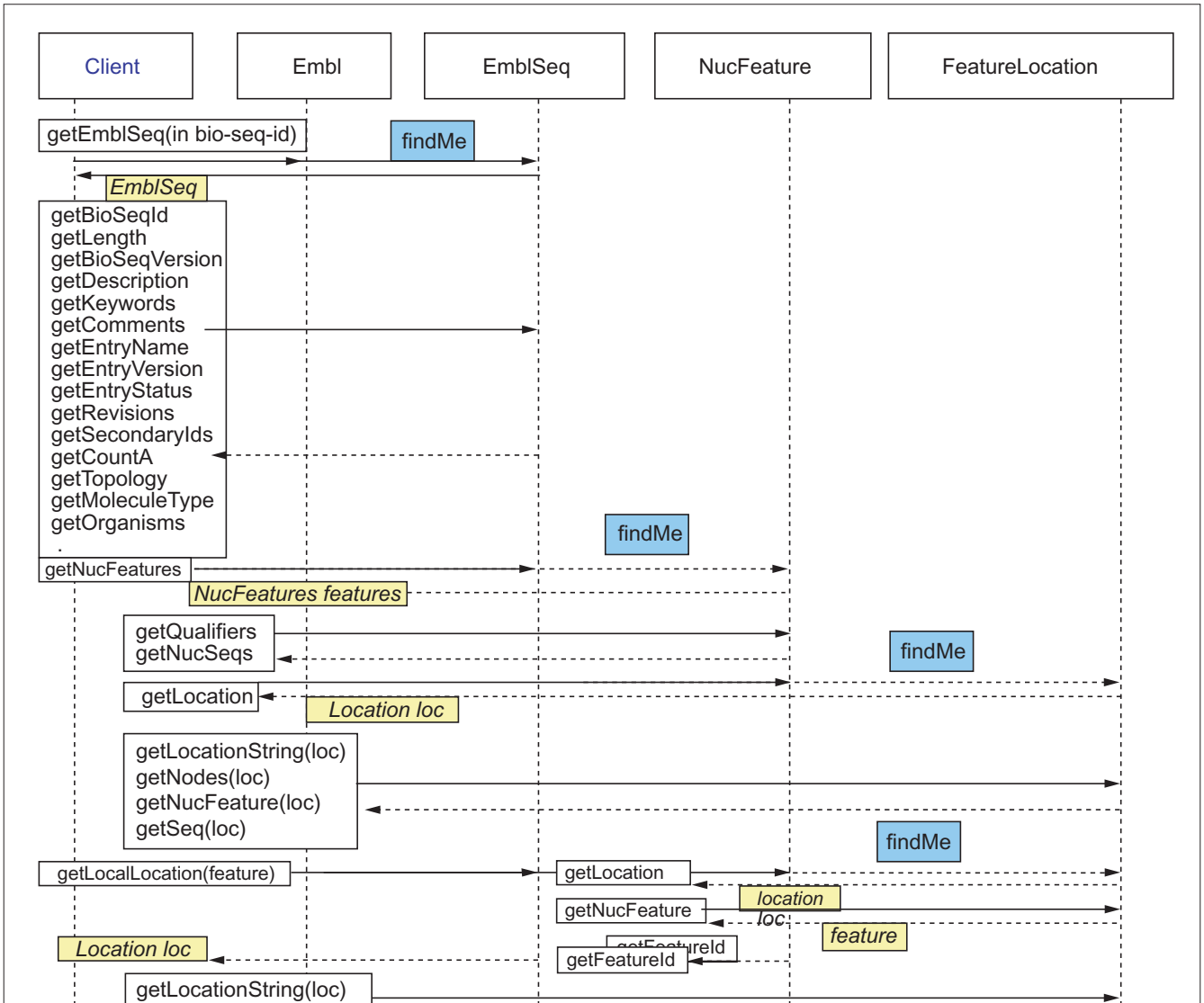

Figure 10
Access to the EMBL database via the CORBA server. The client submits its query with *bio-seq-id* to the *Embl* object, which is a factory object representing the whole database. It invokes the operation findMe provided by *EmblSeq* object, which in turn invokes the loader object. The *EmblSeq* reference is returned to the client. Once the client has the *EmblSeq* object reference, it can then invoke the methods provided by *EmblSeq* to get the sequence information defined in *SeqInfo*. The object attributes are obtained through invoking the methods. Further queries can be made through the invocation to other methods.

One more interesting issue of the evictor pattern is how to choose which object to evict. There are a number of possible strategies, such as least recently used (LRU), least frequently used (LFU), evicting the object with the highest memory consumption (HMC), or using a weighted function that chooses an object for evictor based on a combination of factors (WF). We use a simple LRU algorithm to implement the evictor and prove it is effective.

Accessibility of the EMBL database

When the CORBA server is up and running, a client, which can be developed using any CORBA-compliant ORB on the user's preferred environment and language (for which the ORB is available) at any local or remote machine, can access EMBL data through these objects using an IOR (Interoperable Object Reference) or via a Naming Service. We have published our EMBL server IOR [22] and its naming as

'databases/EMBL/nsdb/Embl', which is registered with the Naming Service [23]. We have also provided a number of demonstration clients for the EMBL server [24].

The client submits its query with bio-seq-id to the Embl object, which is a factory object representing the whole database. It invokes the operation findMe provided by EmblSeq object, which in turn invokes the loader object. The EmblSeq reference is returned to the client. Once the client has the EmblSeq object reference, it can then invoke the methods provided by EmblSeq to get the sequence information defined in SeqInfo. The object attributes are obtained through invoking the methods. Further queries can be made through the invocation to other methods. The access is shown in Figure 10.

From Figure 10, it can be seen that CORBA interfaces to the EMBL database provide: meaningful units of information: objects; encapsulated methods, for example getLength(); interoperation between objects; easy access and distribution of data; and easy to comply with the standard. It provides a basis for developing further biological research tools.

Accessibility to the EMBL database via the EMBL CORBA server can be as fine as any small attribute defined in the EMBL data model. The EMBL CORBA server can also provide a blob object that contains a number of objects. Currently the EMBL server is undergoing a trial with internal and external users. There are increasing numbers of users developing their applications using our CORBA server.

Conclusions

This paper presents a CORBA infrastructure developed at EMBL-EBI. The EMBL object model provides a basis to develop the CORBA server. Employing Persistence™ maps the object model to the relational schema in the underlying Oracle database. To present Persistence with the right relations, views have been used to transform the vertically mapped tables to horizontal ones. Properly built loaders make use of the technique of 'live object caching' and enhance the performance. The evictor pattern is used for memory management. It has been demonstrated that the CORBA server addresses some problems of the flat-file format and provides a solution to accessing and distributing EMBL sequence data. It also provides a flexible and scalable environment for users to develop their applications by building clients.

The future work will include migrating the implementation of the EMBL server to comply with the emerging standard - OMG standard for biosequences. By OMG rules, the EBI, as a co-submitter on the Biomolecular Sequence Analysis (BSA) standard, is obliged to implement the standard. As the BSA standard proposal is not fully compatible with the EMBL IDL specification currently used, care will have to be taken to make this transition as easy as possible for existing clients.

Additional data

The following additional data are included with the online version of this article: The EMBL Nucleotide Sequence Database object model and The EMBL IDL specification.

References

1. **The EMBL Nucleotide Sequence Database** [http://www.ebi.ac.uk/embl.html]
2. **Sequence Retrieval System** [http://srs.ebi.ac.uk/]
3. Henning M, Vinoski S: *Advanced CORBA Programming with C++*. Reading, MA: Addison Wesley; 1999.
4. **OMG, CORBA/IIOP 2.3.1 Specification, 99-10-07** [http://www.omg.org/corba/cichpter.html].
5. Siegel J: *CORBA Fundamentals and Programming*. New York: John Wiley & Sons; 1996.
6. Siegel J: *Corba 3 Fundamentals and Programming (OMG)* 2nd edn. New York: John Wiley and Sons; 2000.
7. Barillot E, Vaysseix G, Achard F, Viara E, Flores T, Rodriguez-Tomé P: *Solutions to the interoperation of biological databases.* Proceedings of the Human Genome Mapping Conference, Heidelberg; 1996.
8. Emmanuel B, Leser U, Lijnzaad P, Cussat-Blanc C, Jungfer K, Guyon F, Vaysseix G, Helgesen C, Rodriguez-Tomé P: **A proposal for a standard CORBA interface for genome maps.** *Bioinformatics* 1999, **15:**157-169.
9. Jungfer K, Rodriguez-Tomé P: **Mapplet: a CORBA-based genome map viewer.** *Bioinformatics* 1998, **14:**734-738.
10. Lijnzaad P, Coppieter J, Flores T, Helgesen C, Slidel T: *CORBA and Molecular Biology*. Addendum to Proceedings of the 12th Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA97), Atlanta; 1997.
11. Parson JD, Rodriguez-Tomé P: **JESAM: CORBA software components to create and publish EST alignments and clusters.** *Bioinformatics* 2000, **16:**313-325.
12. Robinson AJ: **Future directions for providing public access to molecular biology databases and services.** *Eur BioPharm Rev* 1998, 68-76.
13. Rodriguez-Tomé P, Lijnzaad P: **The Radiation Hybrid Database.** *Nucleic Acids Res* 1999, **27:**115-118.
14. Fowler M, Scott K, Booch G: *UML Distilled, Second Edition: A Brief Guide to the Standard Object Modeling Language*. New York: Addison Wesley; 1999
15. **OMG, Unified Modelling Language Specification, UML V1.3, ad/99-06-09** [http://www.omg.org/uml/].
16. **The EMBL Nucleotide Sequence Database Object Model** [http://corba.ebi.ac.uk/models/emblom_doc.html].
17. **Persistence Software Inc** [http://www.persistence.com].
18. **IONA Technologies PLC** [http://www.iona.com/].
19. **The EMBL IDL specification** [http://corba.ebi.ac.uk/EMBL_servers.html].
20. Agarwal S, Keller AM: *Architecting Object Applications for High Performance with Relational Databases*: Persistence Software Inc., San Mateo, CA; 1998.
21. *Orbix C++ Programmer's Guide*. IONA Technologies plc, Dublin, Ireland; 1999.
22. **The EMBL CORBA server IOR** [http://corba.ebi.ac.uk/IOR/Embl.IOR]
23. **The CORBA Naming Service IOR** [http://corba.ebi.ac.uk/IOR/naming.ior]
24. **The EMBL CORBA Clients** [http://corba.ebi.ac.uk/clients.html]