Clustalnet: the joining of Clustal and CORBA

F. Campagne

Department of Physiology and Biophysics, Box 1218, Mount Sinai School of Medicine, One Gustave L. Levy Place, New York, NY 10029-6574, USA

Received on August 24, 1999; revised on December 14, 1999; accepted on January 28, 2000

Abstract

Motivation: Performing sequence alignment operations from a different program than the original sequence alignment code, and/or through a network connection, is often required. Interactive alignment editors and large-scale biological data analysis are common examples where such a flexibility is important. Interoperability between the alignment engine and the client should be obtained regardless of the architectures and programming languages of the server and client.

Results: Clustalnet, a Clustal alignment CORBA server is described, which was developed on the basis of Clustalw. This server brings the robustness of the algorithms and implementations of Clustal to a new level of reuse. A Clustalnet server object can be accessed from a program, transparently through the network. We present interfaces to perform the alignment operations and to control these operations via immutable contexts. The interfaces that select the contexts do not depend on the nature of the operation to be performed, making the design modular. The IDL interfaces presented here are not specific to Clustal and can be implemented on top of different sequence alignment algorithm implementations.

Availability: This implementation is freely available for non-commercial use for a variety of architectures. Clustalnet can be downloaded from http://transport.physbio.mssm.edu/clustalnet.

Contact: campagne@inka.mssm.edu

Supplementary Information: Alignment encoding description and full IDL in the documentation, with the distribution.

Introduction

The Clustal program (Higgins and Sharp, 1988, 1989; Higgins et al., 1992) is frequently used to align biological sequences. The program comes in two versions: Clustalw (Thompson et al., 1994; Higgins et al., 1996) is a command line program that performs alignments, ClustalX (Thompson et al., 1997; Jeanmougin et al., 1998) is an X-window-based implementation that shows, and to a limited extent allows viewing and interaction with, the sequences that are part of the alignment. Recently, the Clustal program, which is freely available, has been distributed as a part of commercial packages. In these distributions, Clustal performs the sequence alignments required by the specific features offered by the package. An example of this kind of interaction is the coupling of a sequence alignment interactive editor with Clustal: the interactive editor helps in the manual refinement of the automatic alignment. Realizing that the interaction between Clustal and different third party programs is useful, fosters the development of the reliable and efficient ‘interaction mechanism’ pursued here.

An additional yet very different, motivation comes from the need to create environments that help the analysis of biological data. An environment can be as simple as a set of programs connected with scripts; yet, to make development, testing and maintenance more simple, the environment could be more elaborate. The key issue is the interoperability of the tools required during the analysis process. A sequence alignment program like Clustal would indeed be a useful tool to add to such an environment.

In this paper we describe the ‘interaction mechanism’ that we have designed and implemented.

Method

The Common Object Request Broker Architecture (CORBA; Object Management Group, 1996) can be seen as a partial solution to the problems we mentioned in the introduction†. CORBA is an architecture that provides means to express how an interaction between a client object and a server object will take place.

The Object Management Group (OMG, http://www.omg.org) is a non-profit corporation that co-ordinates the specification and standardization of CORBA. The Life Science Research Task Force (LSR), part of OMG, issued a Request for Proposal (RFP) on Biomolecular Sequence Analysis in 1998 (OMG, 1998). This RFP requested vendors and implementors of biomolecular sequence analysis software to propose a set of specifications to help

† Readers new to CORBA are invited to consult (Orfali et al., 1997) that proposes a conceptual introduction, and (Orfali and Harkey, 1998) for the basis required for actual development in Java.
the interoperation of software developed for this domain of application. In this context, this article describes an alternative to the Alignment Analysis interface described by the latest response to the LSR RFP (OMG, 1999a,b). For brevity, the proposal to the RFP will be denoted as P99-08-01 when the feature discussed is still present in OMG (1999b), or P99-04-04 when the feature is present in OMG (1999a) but has been removed from the latest proposal. For several reasons the solution described here is better fitted to the common task of protein alignment than is the LSR proposal, these reasons will be explained in this paper.

As specified by CORBA, the interaction between two remote objects is declared in the language Interface Definition Language (IDL). The definition of interfaces to communicate between Clustal (server side) and third-party programs (client-side), and the implementation of these interfaces, constitute a complete solution to the problem.

In contrast to an Application Programming Interface (API), the choice of CORBA allows us:

- to consider scenarios where Clustal (the server) is executed on a machine, possibly a fast machine, and the client, for instance an interactive editor, is used on a graphics workstation (CORBA handles cross-architecture communications transparently);
- to interact with clients implemented in languages different from those in which Clustal is implemented (e.g. Clustal is implemented in C while several interactive editors are implemented in Java)

Figure 1 presents an overview of Clustalnet. ClustalServer is the interface that presents the alignment operation. We tried to minimize the changes that needed to be done to the implementation of Clustal. This allows us to propose Clustalnet updates shortly after a new release of Clustal appears1. For this purpose, we added a new option to Clustalw that runs the executables in server mode. In this mode, Clustalw waits for a client to request an alignment operation. When the alignment request comes, it is honored with the algorithm under the options selected from the command line when the server was started.

**ClustalServer interface**

An interface must be designed carefully, as changes to an interface may compromise the operation of the software that uses it. For this reason, the initial interface must be designed so that incompatible changes need not be introduced in the future. We describe here the motivations that we believed important to be present in a first prototype.

In contrast to P99-04-04 we have designed the interface for the common case of protein sequence alignments described in the introduction, where the alignment is small enough to be represented completely on the client. It might be argued that by doing so, we lose the generality sought by P99-08-01. Actually, applications that need the level of generality defined by P99-08-01 are still to be written. Our purpose here is to design an interface which is simple to implement on legacy tools (alignment engines like Clustal) and simple to use by the implementors of visualization and analysis tools. Implementors of tools that would require more generality could choose to use an implementation that follows the final OMG standard, as soon as one becomes available.

The ClustalServer interface inherits from AlignmentEngine and ContextDependentEngine. This expresses that ClustalServer can perform alignment operations (via AlignmentEngine) and that these operations depend on a context (available via ContextDependentEngine). We define each interface in more detail now.

---

1 We think that it is preferable not to integrate Clustalnet to Clustalw. An integrated executable would require from each end user the installation of the CORBA implementation libraries. This is usually not desirable.
AlignmentEngine interface

The AlignmentEngine interface was designed to minimize the number of methods that must be invoked (remotely) to start an alignment operation on the server and get the result back. When the client interacts with the server, a large number of quick alignment requests are expected. An atomic alignment operation also ensures that the server implementation does not need to handle several internal alignment storages simultaneously\(^3\); the server state depends only on the context in which the operation has been requested. A load-balancing scheme is easier to design in this case. If one maintains a pool of servers configured in the same way, a request from one client can be communicated to any server in the pool.

The definition of the AlignmentEngine interface, in the IDL language, follows.

```idl
exception WrongSequenceNumber {};
interface AlignmentEngine {
    gap_encoding align(
        in sequence_encoding sequences,  
in short numseqs)
    raises (WrongSequenceNumber);
};
```

Encoding of the sequences

The sequences need to be sent to the server. Considering that a representation is usually chosen during the early development of a program for the simplification that it brings to the algorithms that the program must implement, and considering that the client and the server seldom perform the same kind of task, it is unlikely that the client and the server will share the same representation of biological sequences or alignments. Clustal for instance, internally encodes sequences as arrays of integers. The Viseur alignment editor (Campagne et al., 1995–97, 1998) was used to encode sequences as linked lists of residue structures because the program needed to attach a number of properties to each residue of a sequence and to add and remove gaps inside the sequences in a constant time regardless of the position of the insertion.

We introduce here an important difference with the solution described in P99-08-01. We choose explicitly not to model sequences and alignments as CORBA objects. This allows us to pass sequences and alignments as values. We do not use the Object By Value proposal (OMG, 1999c) since this mechanism has not yet been finalized by the OMG and many ORB implementations do not support it.

Because a representation translation is likely to be required, we are free to choose an encoding of the sequences as long as it is simple to convert from the client to the chosen representation, and then from our choice to the server representation. We chose to encode sequences as a unique sequence of characters, as follows:

```idl
typedef sequence <char> sequence_encoding;
```

In this representation, each new sequence is introduced by a "*" character. This can be interpreted by the server as a signal to create a new sequence representation, or to prepare the sequence for the next available one. The residues follow as characters in the order in which they appear in the sequence.

The number of sequences encoded is passed to the align method to allow the server to allocate the storage for the sequences before actually starting to decode. Omission of this parameter would force some implementations to parse the sequence representation twice. These implementations are those that need to know the total number of sequences before they can store a single sequence (the Clustal sequence storage allocation scheme has this limitation). The WrongSequenceNumber exception is thrown when the server determines that the number of sequences in the encoding does not correspond to the number passed directly to the align method. This happens when the client fails to prepare a correct encoding of the sequences, and it should only occur at development time.

As OBV become more widely available, we could make sequence_encoding and gap_encoding object values that have fields to contain the representation we defined here and methods to help its parsing by the client implementation.

The client

According to our ClustalServer interface, the client has to prepare the sequences before requesting their alignment. Once the gap_encoding is returned, the client also needs to interpret the result, in the context of its own internal representations.

Encoding of the alignment

Because the client sent the sequences there is no need to send back the residues as part of the response; so we encode the alignment as a sequence of short integers. The sequence has a syntax that makes it possible to describe the positions and number of gaps to be inserted in the sequences in order to reconstruct the alignment as calculated by the server. The representation is described in full detail in the documentation delivered with Clustalnet.

\(^3\) Clustal does not support the feature; would require important rearrangements to this implementation. To avoid the problem, one would have to block the execution of new alignment requests until the alignment has been read by the client and released. These blocking features are absent from P99-04-04.
Parameter control interfaces

Several interfaces have been devised to address the configuration of the alignment engine, they are shown in Figure 2. These interfaces complement AlignmentEngine so that a client can select the context in which the alignment operation will be performed. The central idea in this design lies in the ContextSelector interface and in its helper Context implementations. ContextDependenceEngine only responsibility is to restrict the control of ContextSelector to a single client at a time, thus ensuring that alignment operations are performed in a predictable context.

Once the client has the control of ContextSelector, it can examine the contexts which are supported by the engine. A context selector allows selection of one context at a time. Once a context has been successfully selected, subsequent invocations of the alignment operation will be performed with the parameters encapsulated by the context. This sequence of operations is illustrated on the interaction diagram (Booch et al., 1998) shown in Figure 3.

Implementation

We implemented the ClustalServer interface on top of the Clustalw program, and also implemented three clients. The first two (a C++ and a Java application) were written to test and demonstrate the use of the server. They are available as part of the distribution to help developers get started with Clustalnet. The third client is a Java package named clustalnet that demonstrates the encoding of the sequences and decoding of the alignment. The crover package (Campagne, 1998) is used to store sequences and the alignment.

Discussion

In this paper, we described a set of CORBA interfaces that can be used by a client to request alignment operations from an alignment server. Our interfaces aim to be simple, efficient and modular.

Sequence and alignment representation

This paper defines different sequence and alignment representations to those proposed in P99-08-01. The reasons for this are summarized here.

The alignment operation requires the residues of the sequences only; all other information is not used. Using a sequence of CORBA references to transmit this information imposes that the client first has to construct these objects before passing them to AlignmentEngine. (As explained previously, it is unlikely that the client implementation relies on a CORBA sequence for its internal storage.) The creation of CORBA objects to be passed as references costs more than the creation
of regular objects because their existence needs to be signaled to the ORB (registered with the Object Adaptor, POA or BOA; Schmidt and Vinoski (1997), Orfali and Harkey (1998)). In contrast, sequence encoding is passed as a value and can be created directly from the client’s internal representation of sequences.

The same rationale applies for alignment encoding. The use of an Alignment interface would oblige the alignment engine implementors to:

- implement the Alignment interface to make the gate between the alignment engine’s alignment representation and the CORBA interface representation;
- register the object to be returned to the client with the ORB;
- return the reference;
- block incoming alignment operations until the client notifies that the alignment object is no longer needed or force implementors to allow for the storage of several alignments;
- destroy the alignment implementation after the client notification (curiously, P99-04-04’s AlignmentAnalysis lacks a method to perform the notification).

Context abstraction
The class diagram shown in Figure 2 presents several differences over the design proposed by P99-04-04.

1. The context is encapsulated in an interface whose implementation is obtained from the ContextSelector (implemented by the actual alignment engine). Therefore, the implementors can choose which context they want to support: the default context

Modularity
Some readers might be surprised that AlignmentEngine does not provide any control over the parameters of the alignment operation. This aspect is being supported by different interfaces in order to make simple implementations that do not require this feature to be feasible with a minimum effort of development. Consider the example where several clients, working independently, use a pool of servers that were identically pre-configured (with gap penalties, matrices, etc.) for a given task. When the clients use only AlignmentEngine, they can interoperate with any server implementation that can perform an alignment operation, including experimental implementations. When the experimental implementation is found to be useful, more work can be invested in implementing the context control interfaces for that engine. This engine can then be used by clients which depend on these extension interfaces.

The parameter control interfaces that we describe are an example of extension interfaces. A third level of extension would include interfaces for load-balancing. They could be built on top of the first two levels.
(parameters specified when the implementation is started), immutable contexts (provide pre-defined parameterization which were well tested for the algorithm implemented by the engine and are always valid), or client editable contexts that can possibly provide the most complete configuration of the algorithms that the engine supports. Editable contexts are responsible for ensuring that the combination of parameters is valid for the given algorithm and implementation. In contrast to the Context interface, P99-04-04 relies on CORBA properties. How does the client know which properties can be set for a given alignment engine? How, and when, are errors in the configuration detected?

2. For P99-04-04, the context (parameters) has to be parsed and set each time align() is invoked. Our solution allows clients to set the context once setContext(context) and then perform any number of alignment operations before they release the server’s ContextSelector.

The context control interfaces we present here are general to any type of operation that need to be configured by the end user. A more general interface can be designed to extend Context and make its parameters changeable by the client. Clients would recognize that a context supports this extended interface through the isEditable() method. The interface for editable contexts is not described here.

P99-08-01 proposes interfaces to carry analysis operations asynchronously. These issues are not covered by the present work.

Performances
A number of factors are involved in the actual performances of our implementation. When comparing with an invocation scheme where the clustalw executable is invoked via the UNIX ‘system’ API, several factors must be considered. They are summarized in Table 1. We do not provide numerical experiments to support this model as this kind of measurement is likely to be sensitive to the hardware, operating system of the client and server, ORB implementation choice, load of the machines and the network at the moment of the tests. As can be inferred from the model presented, Clustalnet is expected to be more efficient than the ‘system’ invocation when seq_size is reduced. This advantage is mainly obtained because clustalw is invoked only once for Clustalnet, while it is invoked #align times for ‘system’. As seq_size increases, the advantage of Clustalnet shrinks because the alignment operation starts to be the dominant factor, and it is constant over the two alternatives. When seq_size is large, the compact representation for the alignment returned by ClustalServer is an advantage over the alignment file format that the ‘system’ alternative has to parse (these alignment files contain the alignment plus the sequences). For some sequence representations, the gaps can be directly inserted in the representation, at low cost compared with the parsing of a file, allocating new storage for the alignment, etc.

Conclusion
Integrating an interactive sequence alignment editor to an alignment program like Clustal can be difficult, especially if one seeks a consensus to be able to target several alignment servers from the same editor. The large-scale analysis of biological data is another process where a simple way to align sequences from a program can be valuable. In this paper we have presented a set of interfaces that cover both these aspects. We implemented a prototype to test the interfaces and were able to carry out our daily work more efficiently. We conducted the development of the interfaces interactively with the construction of the prototype and we believe that this approach deeply influenced several points of the work we present. In this perspective, we believe that no interface specification process can be considered complete until at least one

| Table 1. Factors affecting the performances of the CORBA implementation compared to a clustalw invocation via ‘system’ |
|-----------------|----------------|----------------|
| Operation       | ClustalServer  | ‘system’       |
| Encoding cost   | f(seq_size)    | f(seq_size) + open + |
implementation has been successfully carried out. This work is a contribution in this direction: we freely distribute the Clustalnet implementation to allow for its evaluation by the community.

Acknowledgements
The author is grateful to Dr H Weinstein for useful suggestions regarding this work and to the referees for their knowledgeable feedback. The work was supported by National Institute of Health grants DA09083, and DA12408.

References
Campagne, F. (1998) Conception et développement de systèmes d’aide à la compréhension de données biologiques et moléculaires, application à la modélisation des Récepteurs Couplés aux Protéines G. Thèse de Doctorat, Université de Nancy I, September. (Design and development of systems to help the understanding of biological and molecular data, application to the G Protein Coupled Receptors).