GENOMICS 23, 192–201 (1994)

The Genome Reconstruction Manager: A Software Environment for Supporting High-Throughput DNA Sequencing

CHARLES B. LAWRENCE,*[,1] SANDRA HONDA,* N. WAYNE PARROTT,* THOMAS C. FLOOD,* LIHUA GU,* LAN ZHANG,* MUDITA JAIN,† SUSAN LARSON,† AND EUGENE W. MYERS†

*Departments of Cell Biology and Molecular & Human Genetics, Baylor College of Medicine, One Baylor Plaza, Houston, Texas 77030; and †Department of Computer Science, University of Arizona, Tucson, Arizona 85721

Received August 16, 1993; revised April 5, 1994

A new software system designed for use in high-throughput DNA sequencing laboratories is described. The *Genome Reconstruction Manager* (*GRM*) was developed from requirements derived from ongoing large-scale DNA sequencing projects. Object-oriented principles were followed in designing the system, and tools supporting object-oriented system development were employed for its implementation. *GRM* provides several advances in software support for high-throughput DNA sequencing: support for random, directed, and mixed sequencing strategies; a novel system for fragment assembly; a commercial object database management system for data storage; a client/server architecture for using network computational servers; and an underlying data model that can evolve to support fully automatic sequence reconstruction. *GRM* is currently being deployed for production use in high-throughput DNA sequencing projects. © 1994 Academic Press, Inc.

INTRODUCTION

Using current technology, the target for *large-scale* DNA sequencing is on the order of one million contiguous nucleotide basepairs. However, the primary sequence data are obtained in continuous lengths of only a few hundred bases at a time using polyacrylamide gel-based technology. Consequently, the application of a *sequencing strategy* to the reconstruction of long target sequences from the relatively short primary sequence data is an essential component of large-scale sequencing (Hunkapiller *et al.*, 1991).

Most sequencing strategies attack this problem by breaking the large stretches of DNA into smaller overlapping regions approximately 30,000 nucleotides (cosmid clones) to 100,000 nucleotides (P1 phage clones) in length. Strategies vary by the procedure for obtaining primary sequence data from these shorter regions. In a typical project employing a *random* or *shotgun* strategy, a cosmid clone is randomly fragmented, and the resulting fragments are cloned into an M13 phage sequencing vector. The vector provides priming sites for DNA sequencing adjacent to the inserted clone. If only one end of each cloned fragment is sequenced, then there is no prior knowledge about the relationship of each fragment sequence to the original target sequence. If both ends of a fragment are sequenced, then those two sequences have constrained distance and orientation relationships to each other relative to the target sequence (Edwards and Caskey, 1991). The problem of reconstructing a target sequence from primary data when using a random strategy is usually referred to as *sequence assembly.*

On the other hand, *directed* sequencing strategies may have precise information about the spatial and orientation relationships of sequence fragments to the original DNA sequence. One type of directed strategy employs transposable elements (which contain the sequencing priming sites) inserted at various positions in a target sequence. The position of sequence data generated from priming sites in the transposable element relative to the target sequence is known because the position of the insert is known. (Strathmann *et al.*, 1991). Another directed strategy systematically extends sequencing primers along a DNA molecule to discover new sequence, which then serves as a priming site for subsequent extension. The recent discovery that primers of arbitrary sequence may be generated from a library of hexamers (Kieleczawa *et al.*, 1992) will make this strategy more cost-effective.

Several pilot large-scale DNA sequencing projects that will push the rate of DNA sequencing into the 1 Mb per year per laboratory range have recently been initiated. This rate of DNA sequencing will present new data management and analysis challenges not apparent in smaller projects. Some of the issues that must be addressed in a new generation of data management systems are:

[1] To whom correspondence should be addressed. Telephone: (713) 798-6226. Fax: (713) 798-3759. E-mail: chas@bcm.tmc.edu.

- The processing and flow of data from the sequencing hardware to the finished edited sequence should be automated.
- Effective user interfaces for those tasks that require human inspection or interaction must be developed.
- Alternate user interfaces customized for the role of the specific user (i.e., technician, researcher, manager) should be provided.
- Support for a range of sequencing strategies including random, directed, and hybrid strategies should be provided.
- The probability of error should be assigned to each nucleotide position in the finished sequence.
- Multiple simultaneous users should be able to access and work with the data in a specific sequencing project.
- Transparent access to specialized resources such as computational and database servers should be provided.

Much of the data management research to date has focused only on algorithms and methods for sequence assembly, an important problem for sequencing projects using a random strategy (Staden, 1979; Peltola *et al.,* 1984; Kececioglu, 1991; Huang, 1992; Istvanick *et al.,* 1993; Burks *et al.,* 1994; Parsons *et al.,* 1993; Myers, 1994). Recently, programs with improved editing environments for sequence assembly have become available (Dear and Staden, 1991; Smith *et al.,* 1993).

Our research involves the development of a complete software environment to support large-scale DNA sequencing that addresses the issues outlined above. The result of this project is a software product named the *Genome Reconstruction Manager* or *GRM.* The results of this project to date and a description of the application are summarized in this paper.

MATERIALS AND METHODS

Design strategy. Data management and analysis are a significant bottleneck in the DNA sequencing process. In large part, this can be attributed to the lack of effective integrated software tools and not to any major unsolved technical challenge. The goal of *GRM* is to eliminate the data management bottleneck, and to achieve this its design was guided by three main principles:

1. Automate the data flow as much as is practical and minimize the researcher's interaction with the computer operating system.
2. Provide an effective and intuitive user interface when human interaction is necessary.
3. Generalize the system design so that *GRM* can be easily adapted to different laboratories using various gel-based sequencing strategies.

The design and development team included a project manager, two experienced software engineers, and a molecular biologist. The molecular biologist had several important roles on the team: coordinating our group's interaction with users in sequencing laboratories, developing system requirements, collaborating in the user interface design, and serving as a resident expert in DNA sequencing technology.

System requirements were developed from information gathered by observing the practices in several DNA sequencing laboratories and by interviewing key researchers in those laboratories. We were also influenced by emerging trends in sequencing technology, particularly the emergence of directed sequencing strategies (Hunkapiller *et al.,* 1991; Strathmann *et al.,* 1991; Kieleczawa *et al.,* 1992). Our most significant interaction was with the Sequencing Core directed by Richard Gibbs in the Human Genome Center at Baylor College of Medicine. In addition, Chris Martin and Ed Theil from the Lawrence Berkeley Laboratories Genome Center, Richard Wilson from the *Caenorhabditis elegans* Genome Project at Washington University, and Tom Marr at Cold Spring Harbor Laboratories provided details of the sequencing practices used in projects at their facilities.

GRM was designed and implemented using object-oriented methods. Object technology is a modern paradigm for software development that has reached a level of maturity sufficient to be used to develop production systems. It has now gained wide acceptance in the software engineering community.

We employed the Fusion (Coleman *et al.,* 1994) and the OOSE (Jacobson *et al.,* 1992) object-oriented analysis and design methods to translate system requirements to an object-oriented design. *GRM* system design has also been guided by user-centered design practices (Norman and Draper, 1986) in which the design is driven by the goals of the end users and users contribute to the design process, often by being a member of the development team. User-centered design is facilitated by rapid prototyping, where ideas can be implemented, evaluated, and refined in an efficient interactive cycle.

Fragment assembly algorithms. The problem of reconstructing a target sequence from small, randomly positioned and oriented, error-containing fragments of the original sequence has been called the fragment "assembly" problem (Staden, 1979; Peltola *et al.,* 1984; Kececioglu, 1991; Huang, 1992; Istvanick *et al.,* 1993; Burks *et al.,* 1994; Parsons *et al.,* 1993; Myers, 1994). In *GRM,* the core combinatorial problems of determining potential overlaps between fragments of sequence data and their assembly into contigs have been the focus of work by a subgroup at the University of Arizona. This team has constructed a "C"-language function library that provides a basic "kernel" for solving the problem of fragment assembly. The kernel is written in ANSI standard C following the objected-oriented paradigm. The kernel realizes objects of type *overlap graph, constraint set,* and *assembly* that may be created, destroyed, and manipulated only via routines of the kernel. An object managed by the kernel persists until it is explicitly destroyed. The kernel is designed to interface with the *reconstruction subsystem* of *GRM.* New versions of the kernel may be incorporated with a minimum of additional effort since they do not involve changes at the level of the interface.

Kernel algorithms. At a conceptual level the problem of assembling fragments naturally divides into three phases. In the *overlap phase* each fragment is compared against every other to see whether they share a common subsequence, implying that they were sampled from potentially overlapping stretches of the original strand. The result of this first phase may be thought of as resulting in an *overlap graph* in which each fragment is modeled as a vertex and each statistically significant overlap between two sequences is modeled as a directed edge between them. The second, *layout phase* takes the overlap graph as input and generates a series of alternate assemblies or layouts of the fragments based on the pairwise overlaps therein. A layout specifies the relative locations and orientations of the fragments with respect to each other and is typically visualized as an arrangement of overlapping, directed lines, one for each fragment. The general criterion for the layout phase is to produce plausible assemblies that are as short as possible (an appeal to parsimony), but with the advent of mixed-mode sequencing strategies may also be required to meet an additional *set of constraints.* The final, *multialignment phase* simultaneously aligns the sequences of the fragments in a layout, giving a final consensus sequence as the desired reconstruction of the original strand. These final multialignments are thought of as being the resulting *assembly.* The requirements of each phase and the Arizona subgroup's new algorithmic approaches to each are briefly described in the following subsections.

Overlap phase. Sequencing errors must be accommodated in the search for overlapping fragments and while one may not wish to use the more error-laden data near the end of a gel run for the purposes of multialignment and consensus, its use when detecting overlaps can significantly improve closure probabilities for shotgun projects. The ability of dynamic programming approaches to robustly handle *large* error rates is thus desirable. The difficulty is that such full-sensitivity comparisons can be very time intensive. Using a focusing technique in concert with the Four-Russians paradigm (Wu *et al.*, 1992), Myers has developed a prescreener that conservatively eliminates all pairs of fragments that cannot overlap within a given error rate in time proportional to the square of the number of fragments. This filter is a thousandfold more efficient than traditional dynamic programming algorithms yet equally sensitive as it performs the equivalent computation. Any fragments that are not eliminated are then submitted to a newly developed incremental alignments algorithm (Landau *et al.*, 1994) that efficiently examines every possible overlap to find the statistically most significant. The new incremental algorithm permits substitutions in addition to indels, thus correcting a deficiency of previous work by the Arizona team. For a 40,000-bp project with 500 fragments of average length 400, the set of all overlaps was computed in under 10 min on a 65 SPECint machine. Thus, the one-time computation for the first phase is competitive in terms of time with any heuristic, yet has the advantage of full sensitivity.

Layout phase. Many experimentalists now advocate shotgun sequencing to the point of marginal return and then resorting to primer-based or directed methods for achieving completion or closure. Others advocate approaches involving sequencing only those fragments that do not hybridize (overlap) with other sequenced fragments or sequencing both ends of an insert, all in an attempt to improve on the coverage of pure shotgunning. The impact of these mixed-mode sequencing strategies is that one must now produce the most compact layout subject to a collection of constraints modeling the additional information provided by the enhanced strategy. In light of the additional complexity of handling constraints and the fact that the simple greedy heuristic (Staden, 1982; Peltola *et al.*, 1984; Huang, 1992) for producing layouts tends to work well in most cases, we chose to develop a variant of the greedy algorithm that produces solutions that meet all constraints. Like the basic greedy algorithm, fragments are progressively melded together, where melds are chosen in order of the "degree" of overlap between fragments. But in addition the algorithm rejects a potential meld if it violates a constraint or *will lead to* a violation of the constraints. The idea of constraints has also been explored in the context of genetic mapping (Letovsky and Berlyn, 1992) but the repertoire of constraint types required is different and based on AI inference engines. Our layout algorithm is generative in that it produces a sequence of layouts in decreasing order of "quality." For example, it is important to know whether there is more than one way to put the pieces together, especially if both appear equally plausible.

Multialignment phase. We proceed by producing an initial alignment consistent with all the pairwise alignments of the edges in the layout of the previous phase. This is always possible, is computationally efficient, and, since the error rate is typically less than 10%, produces a very good first approximation. In a second step, a "window" is swept over this initial alignment to optimize the alignment in subregions where the use of global overlap alignments produced locally nonoptimal subalignments. Within the window, the alignment is again the result of merging pairwise alignments, but in this case, in a potentially different order according to the best pairwise alignments between the subsequences within the window. With this window-sweep we empirically find the resulting multialignment to be almost-everywhere optimal, especially when the error rate is less than 5%.

Kernel engineering and interface. The kernel interface realizes three types of objects: overlap graph, constraint set, and assembly. For each object there is a primitive to create such an object, to destroy (deallocate) it, and to read and write it to and from a file. Error conditions detected in any kernel primitive cause a trap to a user-determined control point along with a message and code indicating the error. In this way error handling is left to the discretion of the user of the kernel.

One may create an initially empty overlap graph and then add fragments (vertices) and overlaps (edges) to it. To support version control for fragments, the set of fragments in a graph are partitioned into *classes*, each class representing the set of versions of a given fragment. Only one fragment in a class can be *active*, and it is this fragment that is used in assembly and overlap comparisons. The active fragment can be changed, fragments can be added and deleted from a class, and classes can be added and deleted from the graph. If a user does not intend to support version control, then they can simply place one fragment in each class.

When a fragment is added to a class, one can ask that it either be (1) compared against all other active fragments in the graph, or (2) compared against only those fragments with which a currently active fragment overlaps. If a significant (user-specified) overlap is discovered during the comparisons, then an edge is added to the graph modeling the overlap. To keep overlap graph construction completely open-ended, we provide direct access to our overlap detection routine and even permit one to add overlaps produced by other comparators as "custom" edges in the graph. One can also delete any specific edge from the graph if they wish to override one of the edges automatically created when a fragment is entered. Thus, the kernel gives the user considerable control over the construction and nature of the overlap graph.

One may create constraints of the following types. A *fragment constraint* asserts that a specific fragment be in an assembly or out of an assembly. Thus, one can assemble over any subset of the data. An *edge constraint* asserts that a given pair of fragments either must or must not overlap, where the exact nature of the overlap and orientation of the fragments can be optionally specified. An *orientation constraint* asserts that two fragments must be in either the same or the opposite orientation (i.e., 5′ to 3′ vs 3′ to 5′). Finally, a *distance constraint* asserts that two fragments must be placed within a certain distance range of each other in the final layout. There is a primitive for each constraint that returns a constraint set object modeling the single constraint. Another primitive unions constraint sets together to produce true collections of constraints.

The second and third algorithmic phases for fragment assembly are combined under a single kernel primitive that, given an overlap graph and a constraint set over that graph, produces an assembly object. Actually, the primitive is a generator, in that each successive call produces an alternative assembly. An assembly object contains all the information on fragment layout and multialignment. In essence, it is a collection of contigs and for each contig a representation of the multialignment and consensus for the fragments in the contig. A stick diagram of a layout is considered to just be a view of this multialignment at a coarser level of resolution. There are numerous primitives that allow a user to access the information in the assembly to produce displays of it. There are also primitives that allow one to edit the multialignments of the assembly. Internally, a representation of an assembly is built that is very space parsimonious yet allows efficient browsing. Essentially, only the misaligned characters in the multialignment and the relative positions of the fragments are stored. The *GRM* interface presents its conceptual view of an assembly and how it may be edited by simply translating its higher-level view into a sequence of simple kernel function invocations on an assembly object.

Alternate assembly algorithms. Because the fragment assembly problem is an active area of algorithmic research, the *GRM* architecture is designed to accommodate solutions developed by others. The current version of *GRM* also includes the *CAP* function library developed by Huang (1992). Dr. Huang generously modified the programming interface to *CAP* to make it easier to integrate it in the reconstruction subsystem of *GRM*. Users may select the specific algorithm to be applied at reconstruction time.

Development tools and computing environment. GRM is implemented using Smalltalk 80 (ST80) Release 4.1 and VisualWorks 1.0 from ParcPlace Systems, Inc. Certain modules are written in C and linked into the application using the "C-language Programmer's Object Kit" from ParcPlace. Client/server communications are implemented using ST80 BSD-Socket classes and the TCP/IP protocols. The Gemstone Release 3.2.5 object database management system from Servio, Inc., is used for persistent data storage.

GRM runs on Sun Microsystems SPARCstations (1, 1+, 2, IPX, 10) under SunOS Release 4.1.3 running the X11R5 windowing system. Server processes will run on other networked Unix-based hosts. To date, we have used SPARCstations and a DEC 3000 Model 500S AXP for compute server processes. Contact the author (C.B.L.) for information about availability.

RESULTS

Conceptual Organization of GRM

The role of the *Genome Reconstruction Manager* in the large-scale sequence laboratory is to manage the reconstruction of the target DNA sequence from the primary sequence data generated by the sequencing process. The *GRM* application is conceptually organized around four main activities: project management, sequence preprocessing, reconstruction preprocessing, and reconstruction editing.

Project management. Project management issues include controlling read and write access to project data by individuals, the logical organization of project data, the import of data into the *GRM* application, database transaction management, and managing the flow of control through the application.

Sequence preprocessing. The goal of the preprocessing activity is to identify the part of the primary sequence data that can be safely passed to the reconstruction activity. Primary sequence data may contain several types of artifacts that will affect the quality of reconstruction and that can be detected and annotated prior to that step. These include regions of the data derived from the cloning vector or primer sequence, regions containing unreliable data, and repetitive sequence elements and other sequences that may require the data to be handled specially.

Reconstruction processing. The goal of this activity is to compute the probable ordering and orientations of the primary sequence data relative to the target DNA sequence. The main data available for this computation are the mutual sequence similarities of the primary sequence dataset and any relative distance and orientation constraints that may be derived from the specific sequencing strategy employed.

Reconstruction editing. This activity involves the interactive inspection of the ordering derived during reconstruction processing, the alignment of primary sequence data based on the ordering, and a consensus sequence derived from the aligned columns of primary sequence data. During this activity, the researcher will edit the result by correcting inaccurate data in the primary sequence revealed by the alignment and by overriding the details of the automatically generated alignment and the computed consensus sequence. Regions of poor quality data and missing data that will require the acquisition of additional data in the laboratory may also be revealed. The final result of this activity will be a reviewed and edited consensus sequence representing the sequence of the target DNA.

An Object Model for Sequence Reconstruction

One of the challenges in designing *GRM* was developing an object model for integrating the sequence and reconstruction data. In this section, we outline the key domain objects implemented in the system and then describe the class inheritance hierarchy that supports the implementation.

An `AbiGel` represents a subset of the data obtained from a lane file generated by the Applied Biosystems 373A sequencing hardware. It stores the arrays for the four fluorescent traces, the sequence predicted by the 373A software, and the array aligning the index of bases in the predicted sequence with the position of the prediction in the fluorescent data. An `AbiGel` object is "imported" by reading the contents of a file using a C-function library and assigning values of the data to the Smalltalk object's attributes. The gel object is then stored in the database where it is retained in a read-only form for the duration of the project.

A `GelVersion` is an editable view of an `AbiGel` that stores a version of the original predicted sequence that is editable by human users of the object and by alignment algorithms. The object also allows a subregion to be specified by the preprocessing subsystem, which is subsequently used for reconstruction processing. The `GelVersion` is a persistent object that is stored in the database between sessions. The design of the `GelVersion` will permit multiple editable views of a single `AbiGel` to exist in the system and therefore multiple versions of reconstructions on the same data set.

A `SubGel` is an object representing the subregion of the `GelVersion` that may be used for reconstruction processing. The `SubGel` is a transient object that contains no permanent state and is not stored in the database. However, edit operations may be applied to its sequence as if it were a normal sequence object. These edit operations "pass through" the `SubGel` to its parent `GelVersion` where they are maintained. `SubGel` objects are used by `Contig` and `ContigAssembler` objects.

A `Contig` represents the multiple sequence alignment of `SubGel` sequences as calculated by the `ContigAssembler`. The `Contig` permits edit operations on a column of the alignment and at a position in the individual sequences of the alignment. The `Contig` also stores a `Consensus` object representing the consensus sequence derived from the columns of the alignment. A consensus sequence may be generated following different policies such as "unanimity" or "majority rules" (referring to the nucleotides in a column of the alignment).

A MetaContig represents two or more Contig objects whose position and orientation relative to each other have been deduced by the MetaContigAssembler from a set of Constraint objects.

A Reconstruction object is one of the key organizational objects in the system. It is responsible for storing the current state of reconstruction (all Contig and MetaContig objects) for a project and for coordinating the process of reconstruction with a ContigAssembler and MetaContigAssembler.

A ContigAssembler is responsible for managing the interaction of objects in *GRM* with a fragment assembly algorithm. The ContigAssembler launches an assembly algorithm packaged as a Unix process on a specified network host computer and sets up an interprocess communication path with the remote process. It then coordinates sending sequences derived from submitted SubGel objects to the assembly algorithm and creates Contig objects from resulting assembly layouts.

A MetaContigAssembler is responsible for creating MetaContig objects from Contig and Constraint objects. If conflicting constraints cannot be satisfied, the conflicts are noted and can be made available to a user for resolution.

A ConstraintMap is a type of PhysicalMap from which Constraint objects may be derived. The ConstraintMap is used in *GRM* to represent strategy-specific information that contributes information to computing a correct reconstruction. For example, a project based on a shotgun strategy will often include sequence data derived from sequencing both ends of an insert in a M13 cloning vector. In the final reconstruction, the two individual sequences should appear in opposite orientation and within a distance of each other constrained by the average size of the inserts in the shotgun library. In *GRM,* we use an instance of ConstraintMap to represent the relative position and orientation of two sequences obtained from each end of a M13 insert. Upon request, a ConstraintMap will generate a set of DistanceConstraint, OrientationConstraint, and OrderConstraint objects that are used by the MetaContigAssembler in the construction of MetaContig objects.

Because many of these object classes shared common properties, we were able to develop the inheritance hierarchy shown in Fig. 1. In the figure, classes labeled in italics are abstract classes (classes that define a conceptual object for which no actual instances exist in the system). All classes representing sequences or "maps" relating sequences are derived from the abstract class PhysicalObject, which represents objects having a "physical" extent—in this case a length in units of nucleotides. All classes representing a type of sequence are derived from the Sequence class and all editable sequence types from the EditSequence class. All classes that represent the relative positioning of two or more PhysicalObject instances are derived from

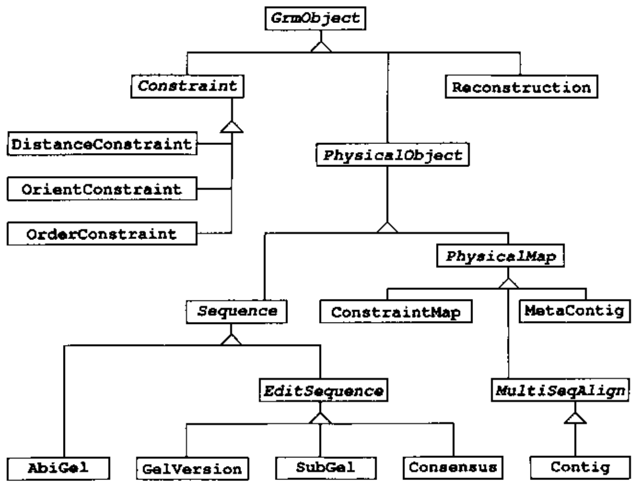


FIG. 1. Class inheritance hierarchy for *GRM* object model. The figure is a diagram summarizing the inheritance relationships for the key object classes described in the text. In the diagram, a class is indicated as a rectangle labeled with the name of the class. Classes labeled in italics are abstract classes (a class for which no instances are created in the application). All other classes are concrete. Classes connected to the base of a triangle inherit attributes and methods from the class connected to the tip of the triangle.

the PhysicalMap class. The base class for most objects in the system is GrmObject, which provides the protocol for assigning a unique identification symbol and a name for an object.

The figure illustrates only a single inheritance hierarchy. However, many of the classes that we developed are functionally the result of multiple inheritance. A good example of this is the Contig class, which directly inherits its ability to represent contiguous overlapping sequences from MultipleSequenceAlignment and its editing protocol from EditSequence.

Because so many of the domain objects in *GRM* have common properties, the use of object class inheritance allowed us to reuse a great deal of code in implementing the design and improves our understanding of the design and ability to maintain the code.

Reconstruction Strategy

There were two key goals with respect to sequence reconstruction that we wished to achieve in this system. The first was to develop an architecture that would allow for alternate fragment assembly computational engines to be used. The second was to take advantage of all information available to assist in computing the correct reconstruction, particularly constraints imposed by the specific sequencing strategy used.

The first was achieved by requiring that a fragment assembly engine to be used with *GRM* would provide a minimum set of standard functions for submitting sequence data and retrieving the resulting assembly data structure. Any such engine meeting the standard (and implemented as a callable function library) can be conveniently packaged as a Unix process with which the *GRM* application can communicate using a standard interprocess communication mechanism. The initial production version of *GRM* has two alternative assembly engines: the fragment assembly kernel described in this paper and the *CAP* engine developed elsewhere (Huang, 1992). The user may select to use either engine at the time she initiates reconstruction processing. Future versions of *GRM* may include approaches to fragment assembly contributed by other algorithm developers.

There are at least two approaches to thinking about how to utilize constraints on a reconstruction imposed by the choice of sequencing strategy. One approach is to compute optimal assemblies that meet all distance and orientation constraints provided. The fragment assembly kernel described in this paper has this capability. A second approach is to consider strategy-specific constraints only after computing an assembly based on sequence comparison alone. The constraints may be used to confirm the computed relationships and to build higher level reconstructions (called *metacontigs*) based on the distance proximity of two sequences present in two unlinked contigs. (For example, using a shotgun strategy it is often the case that both ends of an insert in an M13 clone have been sequenced, but the individual sequences are found at the ends of two different contigs. The two contigs may be assembled into a metacontig by the distance and orientation constraint imposed by knowing that they originated from the same cloned insert.) This second approach is reasonable because the computed similarity that fragment assembly is based on is an unbiased measure of the likelihood that two sequences overlap relative to the target sequence. However, strategy-specific constraints are prone to error since there is usually at least one unautomated step in translating information from the sequencing strategy into constraints that can be imported into the reconstruction software. Thus, solutions that assume the correctness of strategy-specific constraints may be incorrect.

The initial production version of *GRM* follows the second approach and uses constraint information to check assembly solutions and to build metacontigs. Inconsistencies between the computed assembly and added constraints are noted and presented to the user for resolution. However, the assembly kernel provides sophisticated capabilities for handling constraints that may be quite useful for computing reconstructions in difficult sequencing projects. Future versions of *GRM* will provide access to these capabilities so that their utility can be explored in the context of large-scale sequencing projects.

Description of the Application

The following is a "walk-through" of *GRM* for typical usage of the application. The key user interface components and their function will be described, as well as some of the underlying functionality that is transparent to the user. The *GRM* user interface (UI) organization and style is similar to that found in many other widely used software packages that have a graphical user interface, so that it is as natural as possible for a user of *GRM* to move between the *GRM* environment and other applications. In particular, we followed recommendations in the Motif Style Guide (OSF, 1991) unless special requirements of our application suggested that we develop a specialized UI solution.

Project management. To start a *GRM* session, the user enters her login name and password in the *GRM Login Manager*. This action gives authorized users access to the Gemstone database management system and the *GRM* application. The *GRM Application Manager*, a compact UI window with pulldown menus, is then started and remains on the workstation desktop for the remainder of the session. The user initiates work for all other activities from the *Application Manager*. The two key pulldown menus are *Projects* and *Tools*. The former launches dialog boxes for creating new projects and opening, renaming, and deleting existing ones. Once a project has been created or opened, the user may select an item in the Tools menu for importing, preprocessing, reconstructing, or examining a project's inventory. An expanded form of the *Application Manager* also presents useful project status information.

The *GRM Importer* permits the user to import files into the application. Once data is imported into the application, it is treated as a specific type of object and stored in the database management system. Currently three types of objects may be imported: primary sequence data (called *gels* in *GRM*), constraints on the distance and orientation relationships between primary sequence data known from the sequencing strategy used, and diagnostic sequences such as repetitive sequence elements.

The *GRM Inventory* permits the user to list and examine objects currently in the opened project.

As a user moves between different activities in *GRM*, the application transparently commits any changes to the database management system.

Sequence preprocessing. The preprocessing activity is supported by the *GRM Sequence Preprocessor*, which is launched from the Tools menu of the *Application Manager*. The user must select a group of gels to preprocess and the preprocessing parameters. This can be accomplished almost automatically because the application manages groups of gels for the convenience of the user and because the preprocessing parameters may be preconfigured for the duration of the project.

Processing data by groups makes it very easy for the user to move data through the application. The following are some of the groups automatically maintained by the application: *all*, all gels in the project; *last imported*, the last set of gels imported into the project; *unpreprocessed,* the subset of gels that have not yet been preprocessed; *preprocessed-excellent,* the subset of gels that have been preprocessed and judged to be of excellent quality; *reconstructed,* the subset of gels that have been added to the growing reconstruction. The user may also create custom groups using the *GRM Group Editor.*

The preprocessing parameters allow the user to configure how the preprocessor will annotate each gel sequence for quality and the location of specific sequences such as cloning vector sequence or repetitive sequence elements. The parameters may be preconfigured for an entire project or modified for preprocessing a particular subset of all of the gels in a project. The *GRM Preprocessing Configuration* tool is used for this purpose.

Preprocessing is activated by clicking on the *Start Preprocessing* button in the *Preprocessing Manager.* A dialog window that informs the user of the status of the selected group of gels as they are being preprocessed then appears. When processing finishes, the *Preprocessing Manager* displays a panel that summarizes information regarding the annotations added to gels during preprocessing. This panel also displays a list of the individual gels. A specific gel may be selected to launch a tool called the *GRM Trace Editor,* which permits the user to edit individual gel data by modifying the boundaries of the gel region that will be passed on for reconstruction and by editing individual nucleotide positions based on a visual inspection of the quality of the underlying fluorescent sequence data.

Reconstruction processing. This activity is supported by the *GRM Reconstruction Manager* selected from the Tools menu of the *Application Manager.* The goal of this activity is to compute the probable ordering and orientations of the primary sequence data relative to the target DNA sequence.

A *reconstruction* is an object that represents the reconstruction state of some subset of all gels and constraints in a sequencing project. When a reconstruction is created for the first time in a project, the *Reconstruction Manager* presents a panel requesting the user to select the fragment assembly algorithm that will be used in the reconstruction process. *GRM* currently provides two alternative algorithms from which to choose.

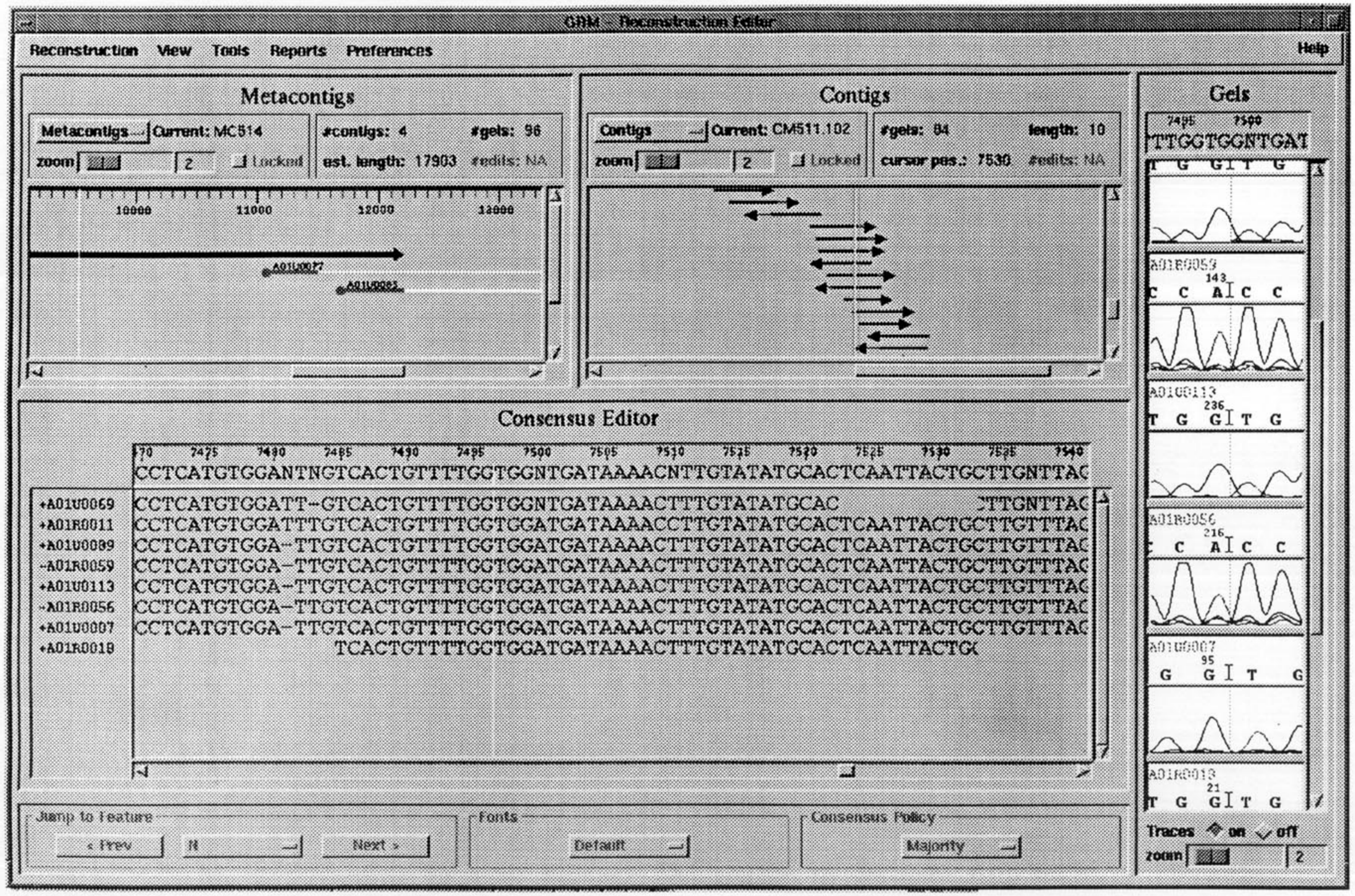
The *Reconstruction Manager* presents a panel to control the submission of gels and constraints for reconstruction processing. The user must select a group of gels and constraints to submit. As with sequence preprocessing, this choice is simplified by selecting a group managed by the application, typically unreconstructed–preprocessed–excellent—the subset of gels that have been preprocessed, judged to be of excellent quality, but not yet added to the reconstruction. The user may alter the sensitivity of the fragment assembly algorithm for detecting significant overlaps between sequences by selecting the level of sensitivity desired from a pulldown menu. The user may also select from a pulldown menu a host processor other than her workstation on which the assembly computation will be performed.

Reconstruction processing is activated by clicking a *Start Processing* button. An informative dialog window that presents continuous feedback regarding the status of the reconstruction process appears. The actual computation is performed by a separate Unix process on the selected host computer whose execution is managed by the *GRM* application. The two processes communicate using standard interprocess communication protocols. When the computation is completed, the dialog window disappears and the *Reconstruction Manager* presents a panel with summary statistics describing the metacontigs and contigs in the reconstruction.

Reconstruction editing. The reviewing and editing of a reconstruction is supported by the *GRM Reconstruction Editor,* which may be launched from the Tools menu of the *Application Manager* or from the *Reconstruction Manager.* This is the most complex UI component of the application and is shown in Fig. 2. During a single session using the *Reconstruction Editor,* the user may examine and edit any metacontig or contig existing in a reconstruction.

There are four key panels in the tool. The upper-left panel is a browser displaying the schematic of a selected metacontig. It shows the relative positions of individual contigs within the metacontig and the constraints that link them together. Within this panel the user may place a cursor on a specific contig to select it. The upper-middle panel is a browser displaying the schematic of a selected contig. This schematic shows the relative positions and orientations of individual gels within the contig and the names assigned to the gels. Within this panel the user may select a column 1-base wide through the contig by placing the cursor with a mouse click. The lower panel is an editor of the multiple sequence alignment computed from the contig layout and of the consensus sequence derived from the columns of the multiple alignment. Within this editor the user may position the cursor on or between columns of aligned nucleotides to select them for editing operations. The panel on the right is an alignment of windows displaying the fluorescent traces from the primary sequence data that correspond to the column of nucleotides selected in the alignment editor. If the depth of aligned sequences is too great for all fluorescent data to be displayed, the user may scroll through the list using a scroll bar. The cursors in all panels are synchronized so that repositioning the cursor in one panel results in the appropriate update of the display and cursor position in the other panels.

The editor includes several convenience features. Different policies (unanimity, majority, etc.) for calculating the consensus bases from the aligned columns are available. The size of the font display in the editor may be adjusted to smaller characters to view a larger



Metacontigs: Current: MC514 | #contigs: 4 | #gels: 96 | zoom 2 | est. length: 17903 | #edits: NA

Contigs: Current: CM511.102 | #gels: 84 | length: 10 | zoom 2 | cursor pos.: 7530 | #edits: NA

```
          70   7475 7480 7485 7490 7495 7500 7505 7510 7515 7520 7525 7530 7535 7540
          CCTCATGTGGANTNGTCACTGTTTTGGTGGNTGATAAAACNTTGTATATGCACTCAATTACTGCTTGNTTAC
+A01U0069 CCTCATGTGGATT-GTCACTGTTTTGGTGGNTGATAAAACTTTGTATATGCAC              CTTGNTTAC
+A01R0011 CCTCATGTGGATTTGTCACTGTTTTGGTGGATGATAAAACCTTGTATATGCACTCAATTACTGCTTGTTTAC
+A01U0089 CCTCATGTGGA-TTGTCACTGTTTTGGTGGATGATAAAACTTTGTATATGCACTCAATTACTGCTTGTTTAC
-A01R0059 CCTCATGTGGA-TTGTCACTGTTTTGGTGGATGATAAAACTTTGTATATGCACTCAATTACTGCTTGTTTAC
+A01U0113 CCTCATGTGGA-TTGTCACTGTTTTGGTGGATGATAAAACTTTGTATATGCACTCAATTACTGCTTGTTTAC
-A01R0056 CCTCATGTGGA-TTGTCACTGTTTTGGTGGATGATAAAACTTTGTATATGCACTCAATTACTGCTTGTTTAC
+A01U0007 CCTCATGTGGA-TTGTCACTGTTTTGGTGGATGATAAAACTTTGTATATGCACTCAATTACTGCTTGTTTAC
+A01R0018             TCACTGTTTTGGTGGATGATAAAACTTTGTATATGCACTCAATTACTG
```

```
7495    7500
TTGGTGGNTGAT
T G G T G

A01R0059
143
C C A C C

A01U0113
236
T G G T G

A01R0056
216
C C A C C

A01U0007
95
G G T G

A01R0013
21
T G G T G
```

zoom 2

region of sequence or to larger characters for increased readability. A *jump* feature allows the user to rapidly move the cursor to the next or previous selected annotation (alignment defect, *Alu* element, etc.) The metacontig and contig panels may be expanded to fill the entire window space to examine large or complicated layouts. *Bookmarks* may be placed in the alignment for quick access to particular regions of a reconstruction during another editing session.

DISCUSSION

The *Genome Reconstruction Manager* is the first software system to address several issues important for supporting high-throughput DNA sequencing. Four capabilities are especially useful:

• Employing an industrial-strength object database management system permits the development of a true multiuser system and increases the reliability of data storage.
• A flexible design for reconstruction permits direct support for a range of sequencing strategies.
• A client/server architecture permits the transparent use of network computational servers.
• The comprehensive object model is designed to evolve to support fully automatic sequence reconstruction.

This project has also allowed us to evaluate the effectiveness of object technology in building software systems for use in genome research. Our experience has been very positive. Sequence reconstruction proved to be a much more complex process than we anticipated at the start of the project. Object-oriented techniques provided us with methods for managing this complexity. The Smalltalk development environment provided us with efficient implementation tools for developing *GRM* with a small team. Much of the *GRM* implementation will be reusable in other related projects. For example, the classes representing sequences, physical maps, multiple sequence alignments, and contigs are not specific to *GRM* and can easily be used in other applications.

Another benefit that we expect from using object technology is the ability to efficiently extend and customize *GRM* as gel-based sequencing technology evolves. An important area for future development not addressed in the current version of *GRM* is the use of sequence accuracy information during reconstruction. In another study we have demonstrated how position-specific error probabilities may be assigned to primary sequence data and described how this information can be used to support automatic alignment editing and consensus determination (Lawrence and Solovyev, 1994). We will also begin development of a companion environment for automatically annotating and visualizing functional information in newly sequenced genome DNA.

ACKNOWLEDGMENTS

This work was supported by a grant to C.B.L. from the Department of Energy Human Genome Program, and by the W. M. Keck Center for Computational Biology. X. Huang generously provided a modified version of his fragment assembly algorithm for use with *GRM*.

REFERENCES

Burks, C., Engle, M. L., Forrest, S., Parsons, R. J., Soderlund, C. A., and Stolorz, P. E. (1994). Stochastic optimization tools for genomic sequence assembly. *In* "Automated DNA Sequencing and Analysis" (J. C. Venter, M. D. Adams, and C. Fields, Eds.), Academic Press, London.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P. (1994). "Object-Oriented development: The Fusion Method," Prentice Hall, New Jersey.

Dear, S., and Staden, R. A. (1991). A sequence assembly and editing program for efficient management of large projects. *Nucleic Acids Res.* **19:** 3907–3911.

Edwards, A., and Caskey, C. T. (1991). Closure strategies for random DNA sequencing. *Methods: Companion Methods Enzymol.* **3:** 41–47.

Huang, X. (1992). A contig assembly program based on sensitive detection of fragment overlaps. *Genomics* **14:** 18–25.

Huang, X., and Miller, W. (1991). A time-efficient, linear-space local similarity algorithm. *Adv. Appl. Math.* **12:** 337–357.

Hunkapiller, T., Kaiser, R. J., Koop, B. F., and Hood, L. (1991). Large-scale and automated DNA sequence determination. *Science* **254:** 59–67.

Istvanick, W., Kryder, A., Lewandowski, G., Meidânis, J., Rang, A., Wyman, S., and Joseph, D. (1993). Dynamic methods for fragment assembly in large scale genome sequencing projects. *In* "Proceedings of the 26th Hawaii International Conference on System Sciences," Vol. 1 (T. Mudge, V. Milutinovic, and L. Hunter, Eds.), IEEE Computer Society Press, Los Alamitos, CA.

Jacobson, I., Christerson, M., Jonsson, P., and Overgaard, G. (1992). "Object-Oriented Software Engineering: A Use Case Driven Approach," Addison-Wesley, Reading, MA.

Kececioglu, J. D. (1991). "Exact and Approximate Algorithms for DNA Sequence Reconstruction," Ph.D. Thesis, Technical Report 91-26, Dept. of Computer Science, U. of Arizona, Tucson, AZ 85721.

Kieleczawa, J., Dunn, J. J., and Studier, F. W. (1992). DNA sequencing by primer walking with strings of contiguous hexamers. *Science* **258:** 1787–1791.

Landau, G., Myers, E. W., and Schmidt, J. P. (1994). Incremental string comparison. Submitted for publication.

Lander, E. S., and Waterman, M. S. (1988). Genomic mapping by fingerprinting random clones: A mathematical analysis. *Genomics* **2:** 231–239.

Larson, S. M., Mudita, J., and Myers, E. W. (1993). "An Interface for a Fragment Assembly Kernel," Technical Report 93-20, Dept. of Computer Science, U. of Arizona, Tucson, AZ 85721.

Lawrence, C. B., and Solovyev, V. V. (1994). Assignment of position-specific error probability to primary DNA sequence data. *Nucleic Acids Res.* **22:** 1272–1280.

Letovsky, S., and Berlyn, M. B. (1992). CPROP: A rule-based program for constructing genetic maps. *Genomics* **12:** 435–446.

Myers, E. (1994). Advances in sequence assembly. *In* "Automated DNA Sequencing and Analysis" (J. C. Venter, M. D. Adams, and C. Fields, Eds.), Academic Press, London.

Norman, D. A., and Draper, S. W. (Eds.) (1986). "User Centered System Design," Lawrence Erlbaum Assoc., Hillsdale, NJ.

Open Software Foundation (1991). "OSF/Motif Style Guide" (Revision 1.1), Prentice Hall, Englewood Cliffs, NJ.

Parsons, R., Forrest, S., and Burks, C. (1993). Genetic algorithms for DNA sequence assembly. *In* "Proceedings of the First International Conference on Intelligent Systems for Molecular Biology" (L. Hunter, D. Searls, and J. Shavlik, Eds.), AAI/MIT Press, Menlo Park, CA.

Peltola, H., Söderlund, H., and Ukkonen, E. (1984). SEQAID: A DNA sequence assembly program based on a mathematical model. *Nucleic Acids Res.* **12:** 307–321.

Smith, S., Welch, W., Jakimcius, A., Dahlberg, T., Preston, E., and Van Dyke, D. (1993). High throughput DNA sequencing using an automated electrophoresis analysis system and a novel sequence assembly program. *BioTechniques* **14:** 1014–1018.

Staden, R. (1979). A strategy of DNA sequencing employing computer programs. *Nucleic Acids Res.* **7:** 2601–2610.

Staden, R. (1982). Automation of the computer handling of gel reading data produced by the shotgun method of DNA sequence. *Nucleic Acids Res.* **10:** 4731–4751.

Strathmann, M., Hamilton, B. A., Mayeda, C. A., Simon, M. I., Meyerowitz, E. M., and Palazollo, M. J. (1991). Transposon-facilitated DNA sequencing. *Proc. Natl. Acad. Sci. USA* **88:** 1247–1250.

Wu, S., Manber, U., and Myers, E. W. (1994). "A Sub-quadratic Algorithm for Approximate Limited Expression Matching," *Algorithmica.* Accepted for publication.