CONF-930114--8

Honda, S., Parrott, N.W., Smith, R., and Lawrence, C. 1993. An object model for genome information at all levels of resolution. *Proceedings of the 26th Hawaii International Conference on System Sciences, Vol. 1* (eds. T.Mudge, V. Milutinovic, and L.Hunter) IEEE Computer Society Press, Los Alamitos, CA. Jan. 5-8, 1993.

An Object Model for Genome Information at All Levels of Resolution

Sandra Honda[1], N. Wayne Parrott[1], Randall Smith[1,2] and Charles Lawrence[1,2]

Department of Cell Biology[1] and Human Genome Center[2], Baylor College of Medicine, One Baylor Plaza, Houston, TX 77030

Abstract

An object model for genome data at all levels of resolution is described. The model was derived by considering the requirements for representing genome related objects in three application domains: genome maps, large-scale DNA sequencing, and exploring functional information in gene and protein sequences. The methodology used for our object-oriented analysis is also described.

1. Introduction

1.1 Motivation for developing a map data model.

Genome data spans a wide range of resolution, from the gross morphology of individual chromosomes to the function(s) of individual nucleotides in an organism. These data and data at all levels of resolution in between (genetic maps, physical maps, genes, etc.) have complex interconnecting relationships.

At Baylor College of Medicine we are developing software systems to support the exploration of genome information at different levels of resolution. Investigators in the Human Genome Center require access to public, community and locally generated data pertaining to genetic and physical mapping progress on human chromosomes. In another project, we are developing a system to automate the assembly and editing of DNA sequences to support large-scale DNA sequencing projects. Finally, we are also developing tools for the efficient exploration of information encoded in genomic DNA sequence and gene products.

We have been motivated by these related projects to develop a seamless software environment in which genome information at all levels is naturally represented by a single underlying data model. We have used the requirements of the three application domains listed above to derive such a model. This report describes the initial results of our work to develop a general object model for genome information. The model is named *GE*nome *M*ap *M*odel or *GeMM*.

FG05-91ER61132

1.2 Motivation for using object-oriented analysis and design

In this report, GeMM is presented as a collection of abstractions which we refer to as objects. An object is a model of an abstract or concrete thing found in the vocabulary of the problem domain and has crisply defined boundaries. Properties of an object include abstraction, encapsulation, identity, state, and behavior (*i.e.* function). An object-oriented system is a software system composed of a collection of objects - each object fulfilling a well-defined role or responsibility of a similar entity found in the problem domain. The description of a system's objects and their responsibilities is called the "object model".

Object-oriented (OO) based development is rapidly displacing all other popular software development paradigms. We have chosen to use OO analysis and design for this project for the following reasons:

- The basic nature of an OO design and its implementation are based upon abstractions found in the problem domain.
- If the domain and problem requirements are well understood, OO designs are typically easy to understand.
- Systems which are easily understood are typically easier to implement, maintain, and enhance.
- OO designs are resilient to changing requirements. This is especially important as changes in technology or new requirements are discovered.

2. Methods

2.1 Object-oriented analysis and design

Typically, object-oriented development refers to the software life-cycle phases: object-oriented analysis (OOA), object-oriented design (OOD), and implementation or object-oriented programming (OOP) [9]. We are using a spiral development model which allows us to itera-

tively revise and refine GeMM by cycling between OOA, OOD, and eventually OOP.

OOA is "the process or study to understand the nature of something or of determining its essential features and their relationships from the perspective of classes and objects from the vocabulary of the problem domain". The product of OOA is a logical object model [1]. The logical object model produced by OOA is passed to the OOD phase where the focus shifts from the aspect of 'what' to that of 'how'. The goal of OOD is to provide an abstract solution (*i.e.*, object solution) that satisfies functional, performance, resource, and other constraints [3].

2.2 Object-oriented analysis and development methodologies

OOA is usually accomplished by following a defined methodology which provides the user with a framework for performing object decomposition and a notation (*e.g.*, graphical, text) for representing an object decomposition. A recent review of OOA methodologies summarizes a variety of competing OOA/OOD methodologies from which a development group may choose [3]. None of the OOA methods we observed is based upon a formal mathematical model.

We have experimented informally with three OOA methods during the development of GeMM:

- Class-Responsibility-Collaborator (CRC) [14]
- Booch's Method [1]
- Object Modelling Technique (OMT) [9]

Each method promotes its own unique philosophy accompanied by techniques and notation for how one conducts the process of developing an object model. OMT and Booch's method are founded on processes involved in developing three complementary representations or models of the problem. A structural model is developed to identify objects in the domain and static inter-relationships such as role, responsibility, associations, *etc.* A dynamic state model is developed to identify the changing nature of object relationships and behavior (*e.g.* a state model). Finally, a functional model is developed to represent the algorithmic nature of the system.

CRC presents a very different OOA process from OMT or Booch's method. Objects are perceived as 'cooperating and collaborating agents' that have 'responsibilities' and can become 'clients and servers' Their interactions are described in terms of 'contracts' [14].

2.3 Model development

We have adopted the CRC method for the initial development of GeMM. Because of its simplicity and anthropomorphic nature, we found it to be the simplest and most natural of the three OOA methods we considered to learn and apply. This has allowed us to quickly develop a common framework to facilitate group-oriented development and mature GeMM to a state that the model can be implemented and evaluated.

This paper summarizes the results of OOA in the genome information problem domain using the CRC method. We first present a brief requirements analysis in narrative form followed by a description of the object model in the CRC method style. We attempt to provide sufficient detail to permit implementation of the model.

3. Results

3.1 Review of objects required for genome mapping projects

3.1.1 Introduction: A core object model that supports integrated genome maps must be able to represent the variety of genome maps that are currently being generated by genome-mapping projects, such as those underway at DOE and NIH-funded Human and model-organism Genome Centers. Genome-level maps fall into three basic categories, genetic linkage maps, cytogenetic maps, and molecularly-based physical maps [2, 6, 10].

3.1.2 Genetic linkage maps: For genetic linkage maps, the map landmarks are polymorphic allele markers, such as restriction fragment length polymorphisms (RFLPs), short tandem repeats (STRs), and variable number tandem repeats (VNTRs). The coordinate system used to measure the distance between landmarks is meiotic recombination frequency, expressed as centimorgan units (1 cM corresponds to an observed recombination frequency of 1% between 2 markers). The resolution of genetic maps is dependent upon sample size -- the total number of meioses observed; if no recombination events are observed between two closely linked markers, the relative order and the distance between the markers will be indeterminate.

3.1.3 Cytogenetic maps: For cytogenetic-based maps, such as those generated by *in situ* hybridization studies, the landmarks are sites of hybridization of tagged DNA probes. The coordinate system is the visible banding pattern along a given chromosome. Markers are defined as being within one band or another (for the human genome, an average band corresponds to 5-10 x 10^6 base pairs), thus map distance is not explicitly defined. Landmarks (probes) within a band however can be ordered 1) based on the ability to visibly resolve two linked hybridization sites (currently as close as ~100 kB using fluorescence in-situ hybridization on interphase chromosomes), or 2) if chromosome breakpoints (*e.g.* as found in somatic cell hybrids or in individuals, such as disease patients, with natural translocations or deletions) exist that physically

separate or delete landmarks.

3.1.4 Physical maps: For molecularly-based physical maps, the landmarks are identifiable sites within a given stretch of (most often cloned) DNA, such as restriction endonuclease cut sites, hybridization sites of complementary probes, chromosome breakpoints, and sequenced regions, and the coordinate system is base pairs of DNA. Low resolution physical maps include the assemblage of overlapping clones of DNA (called contigs), where each clone is, for example, a megabase sized piece of DNA carried by a yeast artificial chromosome (YAC) vector (contigs are also often constructed from cosmid and lambda clones, carrying up to 40 and up to 17 kilobase [kb] size pieces, respectively). In such contigs, the overlap between two or more clones is often defined only by the presence of a shared hybridization site (*e.g.* an STS, defined below) such that the exact location of the shared site in the overlapping clones, and thus the extent of overlap, is undetermined. In some cases one or both ends of a YAC clone are sequenced, creating a unique sequence tagged site (STS), that can then be used a probe to identify additional overlapping YACs, as performed in "chromosome walking" projects. The breakpoints of chromosomal deficiencies, such as those obtained from patients with a genetic disease caused by a gene deletion, can be similarly mapped by the presence/absence of probe hybridization sites and combined into overlapping deletion contigs. The highest resolution physical map is the identification of the sequence of each base of a DNA molecule, with the ultimate map being the complete sequence of all chromosomes of an organism (for humans, 24 chromosomes containing a total of approximately 3×10^9 bp of DNA). Given the sequence, landmarks from lower resolution physical maps, such as STSs, as well as landmarks derived from gene structure/function studies, such as disease gene (exon and intron) boundaries and promoter sites, can be directly placed on a sequence map.

3.1.5 The problem of map integration: Until the sequencing of an organism's genome is complete, comparing, converting, and integrating maps derived by different experimental techniques and with incompatible landmarks and/or coordinate systems, is a major problem in genome mapping efforts (and in the development of systems to represent integrated genome maps). This is especially true when comparing genetic linkage and physical maps since, for example, genetic linkage map distances, measured as meiotic recombination frequencies, cannot be directly converted into physical (bp of DNA) distances. To address this issue, Olsen *et al.* [8] have proposed that current markers be sequenced and converted into STSs, to be used as common landmarks for genome map integration. For example, by sequencing the region around the site of an existing polymorphic allele marker, such as a VNTR, a genetic marker can be converted into a unique STS marker for placement on a physical map.

3.2 Review of objects related to exploring sequence information

The second application domain we have considered is the analysis and exploration of the information contained in nucleotide and protein sequences. The issues that our data model address fall into three main categories:

3.2.1 Relationship of sequences to higher-level maps: One requirement of GeMM is to relate sequences to higher-level genetic and physical maps.

A gene sequence is the highest resolution description of a genetic locus. GeMM should support linking a locus on a genetic map to the region of genome sequence containing the locus, a precise layout of the locus' gene structure (regulatory regions, exons, translation product), and a description of variants of the gene structure, if any, that result in inherited disease.

As described above, polymorphic sequence tagged sites (STS) are short regions of genome sequence that can be placed on both meiotic and physical maps to link them. In addition to supporting descriptions of meiotic and various types of physical maps, our model should support the placement of an individual STS on both types of maps. The STS may be the only known sequence on a cloned segment of DNA (cosmid clone), or a part of an extensive known sequence. A description of the polymorphisms that permit a STS to be placed on a meiotic map should also be supported.

Finally, sequences are directly linked to physical maps of YAC and cosmid contigs which serve as the substrate for DNA sequencing in the human genome project. Our model should support direct linking of sequences to the higher-level physical maps from which they originated.

3.2.2 Representation of non-contiguous genetic components: Another requirement of the model is to provide flexible representation of sequences originating from multiple non-contiguous genetic components. Examples of this include recombinant DNA and both *cis-* and *trans-* spliced RNA molecules. It is important to retain information regarding the origin of each of the components in such a molecule, and to be able to trace back to the parent molecules.

3.2.3 Annotations and derived data: Another requirement of GeMM is to permit the association of annotations and derived data with sequences. Annotations include literature citations, comments, and references to

entries in other databases. It should be possible to associate an annotation with a specific base, a region of sequence, or with non-contiguous regions of the sequence. Sequences are often used as input data for algorithms or methods that produce derived data. Some simple examples of derived data are arrays of values representing average G plus C content along a nucleotide sequence, or hydrophobicity along a protein sequence; the location of matches to certain diagnostic patterns representing functional interactions (*i.e.* promoters); and regions of similarity to other known genes and proteins. Derived data such as this is a type of annotation.

3.3 Review of objects required to support large-scale DNA sequencing

We are interested in developing software support for the process of large-scale DNA sequencing [5]. With current technology, the target for large-scale sequencing is on the order of 1 million contiguous nucleotide base pairs. Current 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. Primary sequence data is obtained from these smaller regions 400 to 700 bases (sequence gel data) at a time by different strategies. One way the strategies vary is in the amount of knowledge that the investigator has about the spatial and orientation relationship of the sequence fragments to the target DNA sequence. The problem of reconstructing the target sequence from primary data is usually called sequence assembly.

This discussion is limited to a description of the conceptual map and sequence-related objects required to support large-scale DNA sequence assembly. This application domain spans a wide range of map resolution from relatively low-resolution YAC maps to high-resolution information related to individual nucleotides. An additional factor that complicates our analysis is that specific process strategies for large-scale sequencing involve some unique objects that are strategy specific.

3.3.1 Data-flow for sequence reconstruction: The main data-flow for the sequence reconstruction process is simply described:

1. Primary sequence gel data is obtained from the sequencing process, typically from an automated gel electrophoresis machine with on-board image analysis capabilities that provide a preliminary interpretation of the data and predicted sequence.
2. The primary data is 'cleaned' to remove characteristic artifacts in the data.
3. The cleaned gel sequence is 'assembled' mathematically to create sequence 'contigs'. The assembly process may use constraint information if prior knowledge about the relationship of fragments to each other is available.
4. A consensus sequence is automatically generated from information containing in the regions of fragment overlap in the contig. The resulting sequence consensus may be manually inspected and edited by a technician.

Despite the relative simplicity of the process, the objects representing primary sequence data, sequence contigs and consensus sequence have sophisticated requirements.

3.3.2 Sequence gel data: One requirement of our model is to represent the primary experimental sequence data. The sequence gel data includes technology dependent and independent parts. The main technology dependent part is the user-accessible gel data prior to 'base calling' by technology-specific software. For example, data from the Applied Biosystems 373A includes a vector of intensities for each of four fluorescence-labeled nucleotides. This data is used in part by the 373A software to predict the nucleotide sequence of the fragment. It is important that the uninterrupted data continue to be associated with the predicted sequence obtained from the gel. At a later stage in the process, the technician will be allowed to override the system's automatic consensus calling and will be assisted by a graphic display of the primary fluorescence intensity data.

Only a portion of the original gel sequence is useful for the assembly step and must be 'cleaned' first. This usually involves removal of sequences from the ends of the sequence. The 5'- end of the gel sequence will contain a short region originating from the sequencing primer that is common to all of the gel sequences and not a part of the target sequence. This must be detected and removed as its presence will corrupt the assembly step. The 3'-end must also be examined as the frequency of error increases significantly. This is usually detected by an increased frequency of ambiguous base calls and the unreliable region is removed. The object representing the gel sequence must remember the original gel sequence and the refined version.

Finally, the 'cleaned' sequence must be inspected for sequences known to be troublesome in the assembly step (repetitive sequence elements) and sequences known not to be in the target sequence (vector contamination). If such sequences are found, this information should be associated with the gel sequence as an annotation.

3.3.3 The sequence contig: The sequence contig is also an object of some complexity. It represents an overlapping subset of the cleaned sequence gels in a project. It must include information regarding the order and orientation of each sequence gel relative to the contig and a reference to each component gel sequence.

In addition, the sequence contig will have a consensus sequence associated with it, and possibly a table of likeli-

hood values indicating a level of confidence in each of the calls of the consensus sequence.

3.3.4 Constraints: Constraints play an important role in the sequence reconstruction process. Especially with 'directed' strategies, knowledge about the order of gel sequences removes most or all of the uncertainty of ordering found in strategies that start with no prior knowledge of relationship of gel sequences (random or shotgun strategy). Even with a random strategy, directed methods are used to finish or 'close' the complete sequence [4]. Types of constraints that must be considered are gel sequences obtained from each end of a M13 clone insert, extended sequence obtained from a sequencing primer chosen from 3'-end of a sequence determined earlier, and sequences obtained from each end of a transposon insert with bi-directional sequence priming sites [11].

Another type of constraint that is often available is a high-level restriction map of the target sequence. This can be used to align contig consensus sequences as assembly progresses. The position of previously-known sequences, and sequences or positions of fragments known to be at the "ends" of the target sequence are useful in reconstructing the sequence.

3.4 Summary of the object model requirements

A core object model that supports the representation of the variety of different map- and sequence-related objects in the three application domains requires:

- the ability to represent a variety of different mappable landmarks
- the ability to represent landmarks with ill-defined boundaries, ordering, inter-landmark distances, and overlapping relationships.
- the ability to represent alternative mappings for the landmarks in a given map region
- the ability to integrate low and high-resolution map information
- the ability to represent maps with different coordinate systems.
- the ability to reference subregions of maps that originate in other maps

3.5 The GeMM Map Object Model

3.5.1 Introduction: Our analysis of the requirements for integrated genome maps, nucleotide and protein sequence analysis, and large-scale DNA sequencing has resulted in the development of a core object model that supports each of these application domains.

Our basic premise is that most and perhaps all information pertaining to genomes can be naturally represented by a 'map' data structure. A 'map' is essentially a collection of 'items' with information pertaining to the arrangement of these items on the map. Items on a map may be as simple as a text comment, or as complex as a nested map hierarchy with arbitrarily complex data items.

We have taken an object-oriented design approach to define a class hierarchy that can represent genetic information from the level of the whole genome of an organism, down to its nucleotide fine structure and data derived by analysis of genetic information. Elements of this design are found in other systems supporting integrated map data.

The core map class hierarchy is:

MapItem
  CoordinateSystem
  Map
    OrderedMap
      SpacedMap
        CoordinatedMap

One class collaborating with the Base Classes is:

MapMeasure

The behavior and responsibilities of each class are described in the following sections.

3.5.2 MapItem Class:

SuperClass: ***Object***

SubClasses: ***Map***, ***SequenceItem***, (many others...)

Description

MapItem is the base class of all objects that can be placed on a 'map'. This class is intended to be subclassed to represent a wide variety of genetic map components and data. For example, subclasses of MapItem will represent polymorphic genetic markers, sequence tagged sites (STS), protein sequence, G+C content in nucleotide sequence, etc. The base class for all maps, Map, is itself a subclass of MapItem creating the possibility of complex hierarchies of maps.

Instances of MapItem have no knowledge about their location on any map, or their relationship to other items on a map. This information is the responsibility of the map containing the item.

An instance of MapItem may be placed on several maps. An important responsibility of each MapItem object is to maintain a reference to all maps containing the item. This permits the convenient retrieval of all maps that have knowledge about a particular item. Genetic maps frequently contain framework markers that are common to many genetic maps as well as unique markers that have special interest to the laboratory developing the map experimentally. This behavior of the MapItem class supports the exploration of related genetic maps through their common framework markers.

We anticipate that there will be many subclasses of MapItem to represent the rich variety of information that can be associated with maps.

Responsibilities

1. Know the size of the MapItem.

   The MapItem object may have

   - size of magnitude zero
   - an exact size in some units
   - a size with some uncertainty

   A collaborating object which is an instance of the MapMeasure class will provide this capability.

   Examples of types of size units that a MapItem object may have are: *nucleotides, amino acid residues, kilobases, centimorgans.*

2. Know all the Map objects that contain this MapItem.

   The Map subclass of MapItem and all of its subclasses may 'contain' one or more MapItems. A single MapItem instance may be on zero or more instances of Map.

   Each MapItem instance must know all of the Map instances that contain it. This capability is provided a collaborating object that is an instance of SequenceableCollection or Dictionary (mapReference). Each time a MapItem object is placed on a Map object, a reference to the Map object is placed in the MapItem object's mapReference. It is the responsibility of the Map object to send a message to the incoming MapItem to initiate this activity.

   When an instance of a Map is destroyed, each of it's MapItem components must be informed that it is being removed from the Map so that the appropriate entry in the mapReference can be cleared.

   A MapItem cannot be destroyed if there are entries in its mapReference.

3. Remember a user supplied 'comment'.

   Each instance of MapItem has the ability to remember a comment in free form text. Comments may be user supplied or generated by a system component that uses the object.

4. Remember a 'title' for the mapItem.

   A 'title' may be user supplied or automatically generated depending on the context. It is intended to provide textual information about the nature of the object in a graphic display of a map.

3.5.3 Map Class:

SuperClass: ***MapItem***
SubClass: ***OrderedMap***
Description

Map is the base class for all maps. A map is a collection of MapItem objects. Because an instance of Map is itself a MapItem, the items on a map may be Map objects. This capability is important in representing integrated genome maps because individual items on a low-resolution map will expand to high-resolution maps.

Another important responsibility of a Map object is registering itself with each of its MapItem objects' mapReference attribute. The Map class provides no information about the order or spatial relationship among MapItems. These responsibilities are left to its subclasses.

In addition to serving as the base class for all maps, an instance of Map object may be used to represent a group of MapItem objects that lie between two markers on a map, when there is no spatial information about the relationship between the members of the group. This situation occurs frequently in genome mapping.

Responsibilities

1. Know the MapItem objects on the map and provide methods to add, delete and access individual instances of MapItems.
2. Know the 'size' of the map.

   The size of instances of the Map class is the number of items on the map, and the size unit type is 'item'. The ability to know size is provided in its super class. Methods allowing access to size may be overloaded in SubClasses that have a different notion of size.

3. Know the number of items on the map.

   All instances of Map and its subclasses know the number of items on the map.

4. Know the number of items on the map and all submaps originating from this root.
5. Know a set of user specified submaps (subset of map items). [This will be a special attribute that is also an instance of Map. This may be the base of a recursive map with the terminal instances specifying the submap or a compound submap. In subclasses of Map, the *subMap* attribute will be an instance of the subclass (OrderedMap, SpacedMap, CoordinatedMap).]
6. Know the components (if any) from other maps that contributed to the construction of a map. [This is a special attribute that is also an instance of Map. The items on this map will usually be submaps from another map. In subclasses of Map, the *components* attribute will be an instance of the subclass (OrderedMap, SpacedMap, CoordinatedMap).]

3.5.4 OrderedMap Class:

SuperClass: ***Map***
SubClass: ***SpacedMap***
Description

The OrderedMap Class adds behavior to the Map Class to know the relative order of items on the map.

Responsibilities

1. Control the ordered insertion and deletion of items on the map.
2. For any instance of OrderedMap or its subclasses on this map, know the orientation of the item relative to the map.

   A map with ordered items has a sense of orientation about items which themselves contain ordered components. It is the responsibility of the map to keep track of the orientation of items on the map. Three possible orientations are supported: NORMAL, REVERSE, and UNKNOWN.

3. Provide access to the map items in their proper order starting at either 'end' of the map.

   An instance of OrderedMap may itself be an item on a higher level ordered map. The higher level map may know the orientation of its items to each other. Thus, ordered access of the items on the lower level map from the higher level map may start at either 'end' of the lower level map, depending on its orientation.

3.5.5 SpacedMap Class:

SuperClass: ***OrderedMap***
SubClass: ***CoordinatedMap***
Description

The SpacedMap Class adds to its superclass the capability of knowing the 'distance' between ordered MapItem objects.

Responsibilities

1. Know the valid distance unit type.

   All distances between instances of MapItem on a SpacedMap object must have the same unit type. The collaborating class MapMeasure provides this behavior.

2. Know the distance between adjacent MapItem objects.

   An instance of SpacedMap knows the distance between all adjacent items on its map. The magnitude of the distance is measured between the 'centers' of adjacent items. Instances of the collaborating class, MapMeasure, represent the distance between items.

3. Know the distance between any two pairs of MapItem instances.

   For some types of SpacedMap objects, the distance between any two items on the map is the sum of the distances between them. However, for other types this is not necessarily the case. For example, the distance (in centimorgans) between markers on meiotic (or genetic linkage)

maps is not linearly additive.

The default behavior of this class is that the 'distance' between any two items is the simple sum of the distance between the items. However, subclasses may implement more complex behavior. For example, a subclass, GeneticMap, may keep track of pairwise distance measures between non-adjacent items.

3.5.6 CoordinatedMap Class:

SuperClass: *SpacedMap*

SubClasses: *SequenceMap, AnnotationMap*

Description

The **CoordinatedMap** Class adds a reference coordinate system to the SpacedMap Class. However, instances of CoordinatedMap are conceptually different than those of its superclass and should be thought of as a map in which the items are pinned to positions in a fixed coordinate system.

The *subMap* attribute (inherited from Map) is an instance of CoordinatedMap and stores definitions of intervals on the map. The *subMap* has the same coordinate system as the higher level CoordinatedMap.

Valid items on a *subMap* are any instance of MapItem or its subclasses. An instance of MapItem precisely specifies a particular interval on its CoordinatedMap object because it has a size and a location relative to the map. A submap with more than one instance of MapItem is compound and usually represents a noncontiguous feature with multiple subregions such as the exons of a gene.

The *component* attribute (inherited from Map) will also be an instance of CoordinatedMap. The items on the map will usually be submaps from another map.

The *subMap* and *component* attributes work together to provide a powerful mechanism to trace genetic material through complex recombinations.

All types of physical maps are represented by CoordinatedMap and its subclasses.

Responsibilities

1. Know the coordinate system of the map.

   This responsibility is fulfilled by the collaborating class 'CoordinateSystem'.

   The instance of CoordinateSystem is an attribute that is also a 'special' item on the CoordinatedMap.

2. Know the location of each item on the map relative to the coordinate system.

   The collaborating class MapMeasure fulfills the responsibility of representing the location of an item on the coordinate system. The location of an item is the location of the 'center' of the item.

   The MapMeasure class also supports uncertainty in the position of an item on the coordinate system.

3.5.7 MapMeasure Class:

SuperClass: *ArithmeticValue*

SubClasses: none

Description

Instances of the **MapMeasure** class represent size and distance on a map.

Instances of this class have some intelligence, and can convert magnitudes between different types of units.

We envision that MapMeasure will be subclassed from a superclass provided in the implementation environment such as the Smalltalk80 ArithmeticValue class. MapMeasure will then inherit methods for performing arithmetic operations and unit conversion.

Private Responsibilities

1. Maintain a table of known unit types specified by the user of the Class.

   There will be a small number of types of units of map distance. The following come to mind in this domain:

   NUCLEOTIDE
   AMINOACID
   KILOBASE
   CENTIMORGAN
   MORGAN
   CARDINALITY (for example, the number of MapItem objects on a map)

   A class variable is responsible for knowing symbols specifying valid unit types.

   The user of the Class will initialize the class with the symbols valid for the particular application.

2. Maintain a conversion table between different map unit types.

   In some cases it will be useful to obtain a magnitude from a MapMeasure object in a different unit type than the one represented by the instance being queried. Since there are exact conversions between certain unit types, the user should be able to ask for the magnitude in units different than the units that a MapMeasure object is assuming. For example, an instance of MapMeasure may be representing length in AMINOACID units. A user of the object may request the magnitude in NUCLEOTIDE units for which there is a simple conversion.

   The user of the class will initialize the conversion tables.

Responsibilities

1. Know a default unit type.
2. Provide access to unit type conversion if available.
3. Know a magnitude.

   The magnitude may be zero or a positive or negative number. The superclass provides knowledge of the unit type of the magnitude.

4. Know the range of the magnitude if uncertain.

   If this value is zero, the magnitude is exact. If non-zero, the magnitude plus or minus the range will be the maximum and minimum values respectively.

5. Answer the magnitude, range, and unit type.
6. Answer the magnitude in a different unit type if requested, and if a conversion table exists.

3.5.8 CoordinateSystem Class:

SuperClass: *MapItem*

SubClasses: none

Description

The **CoordinateSystem** class provides the capability of converting between three different coordinate system views: a user's coordinate system and a 'natural' coordinate system.

The user coordinate system gives arbitrary coordinate values to locations on an instance of CoordinatedMap from the user's perspective (*i.e.* first position is coordinate 1 and increases linearly to 101 plus the length of the map).

The natural coordinate system for CoordinatedMap has a value of 0.0 for the 'origin' of the map, and positive values to the 'right' of the origin and negative values to the 'left' of the origin. This provides as simple mechanism for locating the position of MapItem objects in a map hierarchy on any higher level map. The origin of the map is defined relative to the user coordinate system when an instance of the object is created. It will be most convenient to locate the origin at the exact 'center' of the coordi-

nate system in most cases.

Both the user and natural coordinate systems have the same unit type (inherited from its superclass).

The class can represent both discrete and continuous coordinate systems.

Responsibilities

1. Know the following things about the user's coordinate system:
   - the value of the 'leftmost' and 'rightmost' coordinates
   - if the system is continuous, or has discrete values (sequences have discrete coordinate values)
   - if zero is allowed as a valid coordinate in a discrete system (in displaying nucleotide sequences with coordinates passing from negative to positive, zero is often omitted)
2. Answer the natural coordinate when given a user coordinate.
3. Answer the user coordinate when given a natural coordinate.

3.6 A Data Model for Nucleotide and Protein Sequences

3.6.1 Introduction:

One goal of our work was to design a single object model that supports work with macromolecular sequence information as well as higher level genome information. This would facilitate the development of an integrated genome map database with which the exploration and display of genome information from the highest level to the lowest level would be efficient and natural. We found that subclassification of the CoordinatedMap class results in a straightforward but powerful representation of sequences and related information.

This section describes an extension of the core map object model to represent genetic sequences. The base class hierarchy with Sequence related classes added is:

MapItem
  SequenceItem
    NucleotideItem
    ProteinItem
  Map ...
    CoordinatedMap
      SequenceMap
      AnnotationMap

SequenceItem and its subclasses provide a representation of primary sequence data.

SequenceMap provides a mechanism to construct sequences from an arbitrary number of subsequences. It also provides the interface for end users to interact with genetic sequences and store diverse types of information.

3.6.2 SequenceItem Class (abstract):

SuperClass: *MapItem*
SubClasses: *NucleotideItem*, *ProteinItem*
Description

The SequenceItem class provides storage and retrieval mechanisms for strings representing macromolecular sequences.

Responsibilities

1. Store string representation of a sequence.
2. Retrieve subregions of the sequence.

   Allow retrieval of a sequence or a subregion in the forward or reverse direction.

   Allow retrieval of subregions as if the sequence was circular if requested. It is the responsibility of the requesting object to know if this operation is legal.
3. Provide basic string editing capabilities (but no version control).
4. Provide a lock to prevent further editing.

Responsibilities defined here, but implemented in the subclasses.

1. Know the valid characters in the stored sequence.
2. Validate the characters in a stored sequence.

   Check the characters in the stored sequence against the allowed characters.
3. Maintain character translation tables.

   A character translation table defines a set of characters to be used to convert a stored character to an alternate character. The table should be stored with a symbol tag.
4. Character translate the sequence according to a specified translation table.

3.6.3 NucleotideItem Class:

SuperClass: *SequenceItem*
SubClasses: none
Description

Represent the fundamental atributes of a nucleotide sequence.

Responsibilities

1. Provide retrieval methods unique to nucleotide sequences (retrieve as IUPAC, DNA or RNA sequence and their complements).

3.6.4 ProteinItem Class:

SuperClass: *SequenceItem*
SubClasses: none
Description

Represent the fundamental attributes of a protein sequence.

Responsibilities

1. Provide retrieval method unique to protein sequences (retrieve as amino acid sequence).

   Enforce retrieval only in amino- to carboxyl- direction.

3.6.5 SequenceMap Class:

SuperClass: *CoordinatedMap*
SubClasses: *AbiDataMap*, *ContigMap* ...
Description

SequenceMap and its subclasses provide a mechanism to represent sequences constructed from an arbitrary number of subsequences. Also, because a SequenceMap is a CoordinatedMap, an arbitrarily complex and rich set of information of all types can be stored and accessed as items on the map.

There are many ways to build on the CoordinatedMap class to represent sequence information. The design we describe here is one of several possibilities. An instance of SequenceMap will have 2 new special items (attributes) on its map:

1. *Sequence.* An instance of NucleotideItem or ProteinItem will be used to remember the sequence that the SequenceMap object represents. The size of the item is exactly the size of the coordinate system.

2. *Annotation.* An instance of AnnotationMap will store annotations and data that relates directly to the sequence. The data will be represented by various subclasses of MapItem designed to be specific to the nature of the data.

Figure 3 shows a schematic representation of the SequenceMap class and its superclass.

Attribute: *subMap*

The *subMap* attribute (inherited from CoordinatedMap) may be used to define intervals on a sequence on which to perform analysis. A method of this class creates a new instance of SequenceMap representing the sequence specified by the *subMap*. The new object then becomes the focus for study.

SequenceMap may be subclassed to represent sequences with unique behaviors. For example, in the large-scale sequencing domain, a SequenceRun class will be developed to represent the raw fragment sequence data originating from a automated or gel-based DNA sequencing, and Contig will represent an assembly of many such instances of SequenceRun.

Attribute: *component*

An instance of CoordinatedMap stores references to the components (if any) used in the construction of the SequenceMap object (the child sequence). Each component is an instance of MapItem or its subclasses that is shared with the *subMap* attribute of another instance of SequenceMap (the parent sequence). In both contexts the component represents an interval on the SequenceMap object of which it is a part. However, in the context of the parent sequence it indicates the origin of a part of the child sequence. In the context of the child sequence, it indicates how that component contributes to the sequence.

As an example of how this relationship may be employed, the parent sequence might represent a section of genomic DNA and one of its submaps might represent the location and size of the exons for a mRNA. A method of the SequenceMap class can perform the equivalent of a RNA splice and create a child SequenceMap object representing the exact mRNA sequence. The *component* attribute of the child would indicate how the exons from the parent contribute to the child sequence. This is illustrated in Figure 4.

Another example would be a child SequenceMap object representing a recombinant DNA molecule. The components contributing to the child would have their origin in several parent sequences.

In the example of the spliced RNA, the components exactly concatenate to construct the child sequence. For a recombinant molecule, the components may overlap by a few bases if the they represent restriction fragments with overhanging ends. In fact, there is no restriction on the possible overlapping of subcomponents. If there are conflicting bases in the overlapping subcomponents, the conflicts are represented as ambiguous IUPAC characters in the Sequence attribute of the SequenceMap object.

The *subMap* and *component* attributes work together to provide a powerful mechanism to trace genetic material through complex recombinations.

Responsibilities

1. Know the sequence that this object represents.
2. Provide a repository for derived data and other information pertinent to the sequence.
3. Create new a instance of SequenceMap representing the sequence specified by a submap.

3.6.6 AnnotationMap:

SuperClass: *CoordinatedMap*

Description

The specifications for the AnnotationMap class have not yet been completely developed. The purpose of this class is to represent a wide variety of annotations including text, sequence similarity, the location of matches to recognizers, and various properties of sequences such as average G plus C content for nucleotide sequences or hydrophobicity for protein sequences. Each of the different annotation types will be represented by specific subclasses of MapItem. When a SequenceMap object is 'explored' by applying an analysis method to its sequence, the results will be stored by placing appropriate annotation items in its annotation attribute (an instance of AnnotationMap). The major work to be done is the specification of the various annotation items.

3.7 Application of GeMM to genome maps

Figure 1 illustrates several types of genome-related maps typically used in molecular genetics research, all of

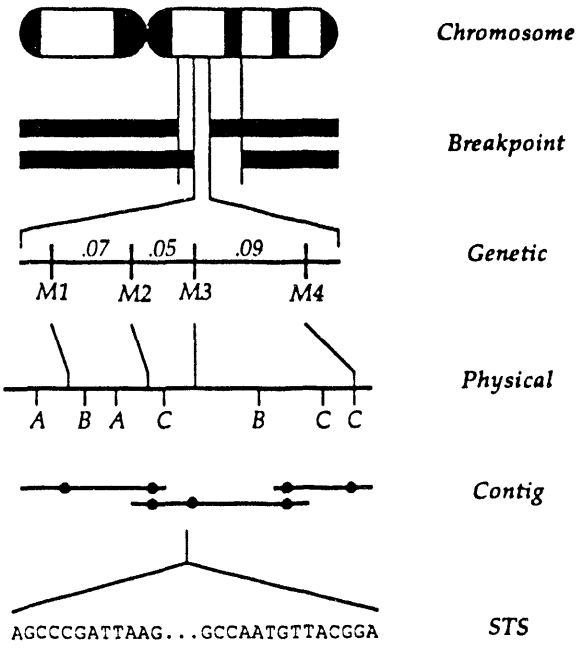

Figure 1

which are addressed by the GeMM Classes. The *Chromosome* is the lowest resolution representation. The figure illustrates the banding pattern used as landmarks in cytogenetic maps. The *Breakpoint* map indicates the location of chromosome deletions found in patients or cell lines relative to the chromosomal banding pattern. Breakpoint mutants are often used to localize genetic markers to subregions of a chromosome. The *Genetic* map illustrates the relative order and distance between 4 genetic markers (M1 - M4) located within a subregion of the chromosome. The distances are in centimorgans. The physical map of this region shows the location of restriction endonuclease sites (A, B, C and D) and the genetic markers (which can also be placed on the physical map because they are STS markers [8]). The *Contig* map shows the relative locations of 3 YAC clones whose overlaps are confirmed by having com-

mon STS markers (circles on the map). One of these markers is expanded to the sequence that defines the *STS*.

The base map classes of GeMM can be easily subclassed to represent each of the specific types of genome maps. The various markers on theses maps are themselves a map or a specialized subclass of MapItem.

4. Implementation of GeMM

The previous sections describe the object model that resulted from object-oriented analysis of our problem domain using the CRC method. We have begun to implement the model for use in a practical system to support the sequence reconstruction process for large-scale DNA sequencing. This system is the *Genome Reconstruction Manager* or *GRM*.

We are implementing GRM using Smalltalk 80 v.4.1 (ST80) from ParcPlace Systems, Inc. ST80 provides a productive working environment for the implementation and delivery of systems using object-oriented methodologies.

GeMM has been implemented in ST80 and extended to represent specific objects in this application domain. For convenience, we implemented the CoordinatedMap Class directly from the Map Class because our application involves only physical maps. The SequenceMap Class has been subclassed to represent several useful classes: 1) the AbiDataMap Cass represents the primary sequence data files; 2) the SequenceContig Class represents partially assembled sequences; 3) the CloneMap Class represents constraints resulting from the closure strategy supported by this application; and 4) the HyperContigMap Class represents the final reconstructed sequence, but with all the working data (SequenceContig, CloneMap, and AbiDataMap objects) associated with it.

While GRM exercises only a part of the GeMM model described in this paper, our experience so far is positive. The CoordinatedMap Class has proven to be a powerful abstraction for representing complex objects related to DNA sequences.

We are very impressed with the productivity provided for this project by the use of object-oriented methodologies, and in particular the relative ease with which we have been able to implement a complex application in the ST80 environment.

Even more productivity gains may be achieved as we develop other applications involving genome and sequence information. We expect the unified GeMM object model to facilitate our development process because of the potential for extensive reuse of code. Another benefit of the model will be the ease with which the applications in different domains can be integrated and exchange information.

5. Acknowledgments

This work was supported by grants from the Department of Energy, the National Center for Human Genome Research, and the W. M. Keck Foundation. A part of the work was also accomplished at the 1992 Gene Recognition Workshop, Aspen Center for Physics.

6. References

1. Booch, G., *Object-Oriented Design with Applications*, Benjamin/Cummings, Menlo, CA, 1991.
2. Caskey, C.T. and Rossiter, B.J.F. "The Human Genome Project: purpose and potential." *J. Pharm. Pharmacol.* 44:198-204, 1992
3. de Champeaux, D. and P. Faure, "A Comparative Study of Object-Oriented Analysis Methods", *Journal of Object-Oriented Programming*, 5(1):[illegible]-33, 1992
4. Edwards. A. and Caskey, C.T. "Closure strategies for random DNA sequencing". *Methods.* 3: 41-47, 1991.
5. Hunkapiller, T., Kaiser, R.J., Koop, B.F., and Hood, L. "Large-scale and automated DNA sequence determination", *Science* 254:59-67, 1991.
6. McKusick, V. A. "Current trends in mapping human genes." *FASEB J.* 5: 12-20, 1991
7. Ohara, O., Dorit, R.L., and Gilbert, W. "Direct genomic sequencing of bacterial DNA: The pyruvate kinase I gene of *Escherichia coli*". *Proc. Natl. Acad. Sci. USA* 86:6883-6887, 1989.
8. Olsen, M., Hood, L., Cantor, C., and Botstein, D. "A common language for physical mapping of the human genome." *Science* 245:1434-1435, 1989.
9. Rumbaugh, J. et al. *Object-oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991
10. Stephens, J.C., Cvanaugh, M.L., Gradie, M.I., Mador, M.L., and Kidd, K.K. "Mapping the human genome: current status," *Science* 250:237-244,1990.
11. Strathmann, M., Hamilton, B.A., Mayeda, C.A., Simon, M.I., Meyerowitz, E.M. and Palozzolo, M.J. Transposon-facilitated DNA sequencing. *Proc. Natl. Acad. Sci. USA* 88: 1247-1250.
12. Staden, R. "Automation of the computer handling of gel reading data produced by the shotgun method of DNA sequencing." *Nucl. Acids Res.* 19:4731-4751, 1982.
13. US Dept. of Health and Human Servicess, US Dept. of Energy. "Understanding our genetic inheritance. The US Human Genome Project: The First Five Years, FY 1991-1995," *DOE/ER-0452P*,1990.
14. Wirfs-Brock, R., B. Wilerson, and L. Wiener, *Designing Object-Oriented Software*, Prentice Hall, Englewood Cliffs, NJ, 1990

END

DATE
FILMED
3/16/94