BIOINFORMATICS

Vol. 23 ISMB/ECCB 2007, pages i337–i346
doi:10.1093/bioinformatics/btm189

A Chado case study: an ontology-based modular schema for representing genome-associated biological information

Christopher J. Mungall[1,*,†], David B. Emmert[2,†] and The FlyBase Consortium

[1]Lawrence Berkeley National Laboratory, Lawrence Berkeley National Lab, Mail Stop 64R0121, Berkeley, CA 94720 and [2]Harvard University, Molecular and Cell Biology: FlyBase, 16 Divinity Avenue, Cambridge, MA 02138, USA

ABSTRACT

Motivation: A few years ago, FlyBase undertook to design a new database schema to store *Drosophila* data. It would fully integrate genomic sequence and annotation data with bibliographic, genetic, phenotypic and molecular data from the literature representing a distillation of the first 100 years of research on this major animal model system. In developing this new integrated schema, FlyBase also made a commitment to ensure that its design was *generic*, *extensible* and *available* as open source, so that it could be employed as the core schema of any model organism data repository, thereby avoiding redundant software development and potentially increasing interoperability. Our question was whether we could create a relational database schema that would be successfully reused.

Results: Chado is a relational database schema now being used to manage biological knowledge for a wide variety of organisms, from human to pathogens, especially the classes of information that directly or indirectly can be associated with genome sequences or the primary RNA and protein products encoded by a genome. Biological databases that conform to this schema can interoperate with one another, and with application software from the Generic Model Organism Database (GMOD) toolkit. Chado is distinctive because its design is driven by ontologies. The use of ontologies (or controlled vocabularies) is ubiquitous across the schema, as they are used as a means of *typing* entities. The Chado schema is partitioned into integrated subschemas (modules), each encapsulating a different biological domain, and each described using representations in appropriate ontologies. To illustrate this methodology, we describe here the Chado modules used for describing genomic sequences.

Availability: GMOD is a collaboration of several model organism database groups, including FlyBase, to develop a set of open-source software for managing model organism data. The Chado schema is freely distributed under the terms of the Artistic License (http://www.opensource.org/licenses/artistic-license.php) from GMOD (www.gmod.org).

Contact: cjm@fruitfly.org or emmert@morgan.harvard.edu.

*To whom correspondence should be addressed.
†The authors wish it to be known that, in their opinion, the first two authors should be regarded as joint First Authors.

1 INTRODUCTION

1.1 On the need for standardized database schemas

Organism-specific genome databases are expertly curated repositories of data and knowledge concerning a particular biological species, or a collection of closely related similar species. These biological databases are typically (but not always) implemented as relational databases that encode their domain model using the relational model. A relational database requires a data base management system (DBMS) to access and update data. Data housed in a database must be modeled according to a *database schema*, a computable description of the data domain, expressed mainly as table definitions. Data modelers, in conjunction with domain experts, design database schemas. Users interact with the database via software applications and user interfaces (often via another layer of indirection, i.e. an intermediate, *middleware* layer). The design and implementation of database applications is time-consuming and labor-intensive. Furthermore, when database applications are constructed to work with a particular schema or set of schemas, changes to the database schema may dictate reciprocal changes to this software. All of this makes *schema evolution* a costly affair. From this it would seem to follow that a small number of stable schemas would be favored over a plethora of rapidly evolving schemas, and yet the latter is more common in bioinformatics. Why is this the case?

1.1.1 New knowledge Schemas must evolve to cope with changes in requirements. Most critical are the changes in the nature of the underlying data, which is constantly accruing and evolving. The nature of biological data has expanded tremendously over time, ranging from classical genetic studies performed a century ago (Morgan, 1907), to present-day genome-scale molecular knowledge. For example, a database schema built around the one-time central dogma of 'one gene codes for one enzyme' (Beadle and Tatum, 1941) would be considerably simpler than a schema that accurately represents our present understanding of the complexities of genetic information transfer. As our understanding of the natural world changes over time, the requirements must necessarily change as well.

1.1.2 New experimental techniques Concomitant with our accrual of biological knowledge, are the advances in the methods and materials we use to gain this understanding. These rapid technological changes place additional requirements on the schema. During the short time that genetic

databases have been in existence we have seen experimental techniques expand from physical mapping and PCR; to sequencing of whole genomes; to modern high throughput technologies for microarray and proteomics analysis: all of which place increasing demands on the database schema that must represent these.

1.1.3 Different organisms A wide variety of species are used in research because each offers unique leverage to explore certain aspects of life. The taxonomic variance of biological properties, along with the different experimental methods utilized in these species, add another dimension to the requirements. Any given organism is selected for a research project based on its utility in answering different questions, and this has made it historically difficult to create a species-blind database schema.

1.1.4 Acceptability Coupled with the innate complexity of the data; the changing requirements as science and technologies progress; and the variability between research projects, the design of stable, shared schemas that are acceptable to a wide variety of different projects is a challenging task. Even within the realm of model organism database projects, the historical tendency has been for each project to design their own schema *de novo*, or in some cases to start with an existing schema and customize it to satisfy a different set of requirements. Such customizations inevitably lead to divergence, loss of interoperability and duplication of effort.

Because these factors make it difficult to create schemas that are stable, schemas are constantly evolving with concomitant high costs in software maintenance. The challenge of biological database design is how to keep pace with a moving target.

1.2 Existing approaches to biological schemas

As might be expected, there are a wide variety of approaches in designing schemas for biological databases. Some of the more notable schemas, with which we have direct personal experience, include: ACeDB, ArkDB, Ensembl, Genomics Unified Schema (GUS) and Gadfly (Mungall *et al*., 2002). ACeDB (A *C.elegans* Database) was one of the first model organism databases. It was built for *Caenorhabditis elegans* (Durbin and Theirry-Mieg, 1994) and is actually a DBMS that follows a hierarchical rather than relational model. ACeDB was adapted for use in a number of model organism projects (as well as projects not related to biology at all, testimony to its flexibility). The ArkDB schema (Hu *et al*., 2001) was created to serve the needs for the subset of the model organism community interested in agriculturally important animals. It has been successfully used across different species by different communities, but is rarely used outside the agricultural community. The Ensembl database system and schema was initially constructed to analyze the newly sequenced human genome (Hubbard, 2002) and serve the results to the scientific community. It has since been adopted by other groups and is used for a large variety of (primarily chordate) species. Its focus has also expanded, and now Ensembl includes a variety of federated databases accessible through the EnsMart. Like the other databases, GUS was specifically built for transcript analysis, and to serve the needs of the Plasmodium research community, and has been extended to serve additional communities. GadFly was designed to serve as a repository of *Drosophila* genomic annotations, but was also used to hold honeybee and *Anopheles* annotations.

1.3 Ontologies and terminologies

Chado differs from the schemas mentioned above by the centrality of ontologies and terminologies as a core component. Chado uses ontologies not just for annotation of biological entities, but also as a schema-wide *entity-typing* and *entity-relationship-typing* mechanism. This methodology of *ontology-driven design* is explored in this article. We contend that it is the key to the success and flexibility of Chado and of its adoption by a wide variety of research projects. In other schemas with which we have experience, typing of the data is enforced at the relational layer. In Chado, in contrast, data typing is driven by ontologies in the controlled vocabularies module, and this makes it possible for the same schema and application to be reused and to evolve over time.

2 SYSTEM AND METHODS

The Chado package uses postGreSQL and Perl. In addition to the Chado DDL (Data Definition Language), installation requires three additional Perl packages: bioperl-live, go-perl and DBIx::DBStag. To install on Fedora Core 1-5, OS X or CentOS 4 you may use the RPM packages for installing Chado, and its prerequisites, provided by Allen Day (http://biopackages.net). Otherwise installation requires checking out the Chado package via anonymous CVS and performing a series of command line operations. Instantiations of Chado in Oracle or mySQL idiom are also available.

3 DESIGN APPROACH

Because Chado makes extensive use of ontologies (also known as controlled vocabularies[1]) as a means of *typing* entities in the schema, and as *metadata* for extensible data properties, an appreciation of the fundamentals of ontologies and how they are coded in the Chado schema is required. The rationale for this approach is 2-fold. It addresses both the significant issue of constantly evolving requirements and provides support for *reasoning*. An ontology is a representation of the different types of entity that exist in the world, and the relationships that hold between these entities. Examples would be the anatomical type 'eye' or the process type 'cysteine biosynthesis'. These types stand in certain relationships to one another; for example, 'eye is_a sense organ' or 'ommatidium part_of compound eye'. The relationships in an ontology can be represented as a graph (often, but not always a directed acyclic graph, or DAG). The OBO relationships paper by Smith *et al*. in 2005 provides a detailed treatment of relationship types in biological ontologies. Of particular interest to Chado is the relation, which specifies a subtyping relationship between two terms or classes. It is the relations that exist between the types in the ontology that supply a means of supporting reasoning.

[1]In fact there are crucial differences between ontologies and vocabularies. However, not everyone agrees on what these are. For the purposes of this article it is simpler to gloss over these differences.

3.1 Chado querying exploits ontological relations

Determining the answers to simple queries involves, broadly speaking, traversing a DAG defined by the ontology along both 'is_a' and 'part_of' edges. For example, the sequence ontology (SO) (Eilbeck *et al*., 2005), a collection of sequence feature types, is used for typing features in the sequence module of Chado. Tables, such as the **feature** table, define a foreign key column to indicate the specific type or class of entity for each row in that table. In addition to typing features, all relationships between features are also consistently typed, using the relationships defined by the SO. For example, SO has a part_of relationship between the type 'exon' and the type 'transcript'. According to the definition of part_of in the OBO relations ontology, this means that *all* exons are necessarily part_of *some* transcript at some point during their existence. To answer the query, 'find all exons that contain both CDS and 5' UTR', may involve selecting all rows in the feature table that are either of type '5_prime_UTR' or of type 'CDS', that are both also 'part_of' the same row in the feature table which is of type 'exon'. Implementing these 'transitive closure' queries can be difficult using many existing implementations of database query languages, so Chado allows for precomputation of these inferences. These SO relationships are uniformly applied, that is, a single relationship type between features always implies the same thing. In cases where a new relationship type is needed, one that is not yet present in the SO (for example 'duplicate' for relationships between chromosome aberrations and genes), the normal process of discussion and resolution takes place to assure this consistency.

3.2 Generic schemas

The combination of the type_id column and the **is_a** relationship gives Chado a data sub-classing system, beyond that possible with traditional SQL database semantics. Chado uses the same table for all different kinds of feature, and uses the SO as a typing system. This same strategy, a single table that is typed according to an ontology, is repeated throughout the schema. Chado balances the strengths and weaknesses of this generic schema approach using a system of layered compliance, a novel approach to relational database typing. The different layers can be seen as akin to a protocol stack in network communication systems.

At the bottom is level-0 compliance, relational schema compliance. As in standard RDBMS implementations, the Chado relational schema defines table structures, foreign keys, uniqueness and other constraints. An example of a constraint enforced at the relational level is a constraint on feature locations, which ensures that the start position of a feature is less than the end position (**fmin** < **fmax**, see SQL). The capabilities that are built into the DBMS require that all data conform to some schema, and automatically enforces such relational constraints, thus it is impossible to have a Chado instance that is not compliant with some version of the schema. However, it is still possible to talk of two versions of the schema being non-compliant with respect to one another, or to talk of one instance of Chado being level-0 backwards compatible with respect to another.

The next level is level-1 compliance, based on ontologies. This means that all values in the type identifier column in the database refer to some subtypes of the official Chado base types. For example, all sequence feature types must be kinds of sequence features; there must be a path over the 'is_a' relation from the feature type to the SO type 'locatable_sequence_feature'. Likewise, all sequence feature property type identifiers must refer to a type that can trace an 'is_a' path back to the type 'feature_property' in the OBO feature property ontology.

Because the standard relational model does not handle subtyping, we call these extrarelational constraints. These constraints can either be encoded in external software and applied as part of a data validation check, or in the DBMS as rules and automatically enforced. If the constraints are encoded in external software such as middleware, then all applications or tools, which are capable of modifying the database, must interface the database through this middleware layer; and this can be overly restrictive on software development. Constraints encoded in the DBMS as rules can be enforced via database triggers, and this is preferred for ensuring data consistency, but it inflicts a data update performance penalty.

4 IMPLEMENTATION

Chado is implemented as an RDBMS, and as such, consists primarily of a collection of table definitions, each of which corresponds to a high-level category of entity. Above these tables, Chado is logically partitioned into different modules, each of which model distinct domains of the data. Modularity (or encapsulation) is a fundamental principle in the design of any large software or information system. It reduces complexity and interdependencies, in contrast to a monolithic approach. Whilst Chado modules are not completely autonomous, and there are some intermodule dependencies, these are minimized. Each module only 'exposes' a subset of its tables to other modules. This modular system allows different local Chado installations to plug in their own modules to extend the schema, or to replace existing modules with their own customized ones.

Five 'core' modules are required by all Chado installations. Beyond these core modules, only the subset of the modules that are required for a project are installed; with the caveat that due to the aforementioned interdependencies, sometimes use of one module will entail the use of another. The five core modules cover: general usage, such as database cross-references; publications and citations; auditing; controlled vocabularies (ontologies); and sequence features. That biological sequence features are core to any Chado implementation is a reflection of the fact that in Chado, data is tethered to the genome similarly to how it is organized in nature. As such, in Chado, sequence features are similarly used for relating genetic, phenotypic and functional data, and these five cross-cutting modules are applicable to all biological domains.

The optional Chado modules cover a diverse range of domains across molecular biology. There are modules in Chado for comparative analysis, expression studies (both microarray and *in situ*), mapping data, phylogenetics, genetics, clone libraries, organism taxonomies, phenotypes and personal address information.

The central module is the **sequence** module, which models biological *sequence features*. Sequence features include the genetically encoded entities such as genes, gene products, exons, regulatory regions and other heritable genomic entities. Sequence features are central to the Chado model and are the focus of this discussion; only when the Chado model of features is understood can one move on to other kinds of data such as expression and genetic data.

4.1 Core modules in Chado

4.1.1 General usage module The primary purpose of this module is to provide data entities with stable, global, unique identifiers, although it also contains other schema metadata tables. In Chado, all identifiable data entities have tripartite identifiers, consisting of a database name, an accession, together with an optional version suffix. In most Chado instances, the version is seldom used, so this is effectively a bipartite identifier.

Each identifier is stored as a row in the **dbxref** table, together with a column linking it to the database name, which is stored in a separate table. A database name uniquely identifies the authority responsible for a particular ID-space. Keeping the database name in a separate table ensures that the schema retains its commitment to normalization (so there cannot be two distinct databases called 'GO' in any single Chado instance). The accession (plus version) must be unique within an ID-space (enforced by the schema). Thus there can be two accessions '0008045', but there can only be one data artefact identified as 'GO:0008045'. All stable identifiers are stored, whether or not they refer to external entities. Chado does not have an explicit notion of a data entity being external. Database names may be associated with URIs for compatibility with other identifier schemes (e.g. LSID, Clark *et al*., 2004). Entries in other tables refer to entries in the **dbxref** table by means of *foreign keys*.

4.1.2 Publication module Data provenance is of central importance in any curated database. In Chado, the **pub** table, in the module of the same name, handles provenance. The name **pub** indicates the most common use for this table, publications. However, the scope of this table is not limited to published documents, in Chado terms, a publication is any attributable source of data, including personal communications and database analyses.

4.1.3 Audit module This module is autogenerated from the database schema itself. For each table in the database, there is a set of trigger functions which populate a single auditing table, called 'audit_chado', such that when there is an update or delete in the parent table, there is an insert of the unique key for the old record into the audit table with a time stamp and an identifier for the user who did the commit.

4.1.4 Ontologies and controlled vocabularies module Ontology and controlled vocabularies are integral to Chado, enabling the generic schema and subclassing, and hence the CV (controlled vocabulary) module occupies a prominent place in the schema. In Chado, each type is represented as an entry in the **cvterm** table. The **cvterm** table is also used for representing all types of relations. Links between types are represented by entries in the **cvterm_relationship** table. Types can have synonyms and definitions (specified using both natural language and machine interpreted logical definitions). The Chado schema CV module represents these and related data.

4.2 Sequence feature module

The **sequence** module, more particularly the **feature** table in this module, is central to Chado sequence data management. Chado defines a feature as a region of a biological macromolecule (i.e. a DNA, RNA or a polypeptide molecule) or an aggregate of regions on this polymer. As the term is used here, 'region' can be the entire extent of the molecule, or a junction between two bases. Features are typed according to the SO, may be localized relative to other features, and may stand in certain relations to other features.

4.2.1 Features and sequences In Chado, all genetically encoded or transmitted entities, including genes, transcripts, proteins, alleles and so on, are modeled as entries in the **feature** table of the **sequence** module. Absolutely no distinction is made between a feature entity and a sequence entity, they are considered one and the same. Entries may optionally have DNA or amino acid residue sequence attached, but attaching residues is not mandatory because it is sometimes necessary to create feature entries for entities whose sequence is currently unknown (such as a gene that is identified only through traditional genetic techniques, or a cDNA that has only been partially sequenced). This is a crucial aspect of the Chado design, and is in contrast to other schemas and common bioinformatics formats, where features are inherently artifacts with mandatory placement on some external sequence coordinate system. It follows that, while a feature may or may not have sequence, every sequence is a feature: it is impossible to store a sequence in Chado except as a feature. This makes Chado different from almost all other schemas used in genomics, in that its design was chosen to reflect both the biological reality and practical considerations—Chado does not need to assign separate identifiers for sequence and feature entries in the database.

4.2.2 Feature location Sequence features are typically localized using a coordinate system. Chado uses a relative localization model: all feature localizations must be relative to another feature. Features (e.g. exon) hold a relationship to a location, i.e. coordinates, which itself holds a relation to a source feature (e.g. chromosome). Locations are stored in the **featureloc** table.

A feature may have zero or more **featurelocs**, although it will typically have either one (for features of which the location is known) or zero (for unlocalized features such as chromosomes, or for features for which the location is not yet known, discovered using classical genetics techniques). Multiple **featurelocs** are used to localize alignments. A **featureloc** is an interval in interbase sequence coordinates (Fig. 1), bounded by the **fmin** and **fmax** columns, respectively, representing the lower and upper linear position of the boundary between bases

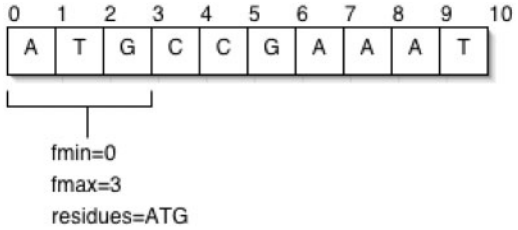

Fig. 1. Chado uses the interbase method to manage sequence length. Counts begin at zero, and the space between two bases is what is counted. In this example, the codon for Methionine begins at 0 and ends at 3.

or base pairs, with directionality indicated by the **strand** column.

Other non-sequence-oriented kinds of localization (such as physical localization from *in situ* experiments, or genetic localizations from linkage studies) are modeled outside the sequence module, for example, in the **expression** or **map** modules, but a discussion of these is beyond the purview of this paper.

Interbase coordinates were chosen over the more commonly used base-oriented coordinate system because they are more naturally amenable to the standard arithmetic operations that are typically performed upon sequence coordinates. This leads to cleaner and more efficient database coding logic that is arguably less prone to errors. Of course, interbase coordinates are typically transformed into the more common base-oriented system used by BLAST reports and henceforth prior to presentation to the end-user. As mentioned earlier, the Chado schema includes a constraint which ensures that **fmin <= fmax** is always true–any attempt to set the database in a state which violates this will flag an error.[2]

The **featureloc** table also holds the **srcfeature_id**, that is, the foreign key referencing the feature that its coordinates are relative to. There is nothing in the schema prohibiting localization chains; for example, locating protein domains relative to polypeptide(s) that in turn may be localized to their respective transcripts, or locating an exon relative to a contig that is itself localized relative to a chromosome (Fig. 2). The majority of Chado database instances will not require this flexibility; all features are typically located relative to chromosomes or chromosome scaffolds. Nevertheless, the ability to store such localization networks or location graphs are particularly useful for unfinished genomes or parts of genomes such as heterochromatin (Hoskins *et al*., 2002), in which it is desirable to locate features relative to stable contigs or scaffolds, which are themselves localized to an unstable assembly to chromosomes or chromosome arm scaffolds.

We will now present a short formal treatment of the properties of these hierarchies of localization using graph theory. This treatment can be ignored for the purposes of understanding the basics of managing sequence data in the Chado schema; the end-user of the database will be entirely

[2]This constraint may be relaxed if the intent is to model circular genomes such as those found in some bacteria.

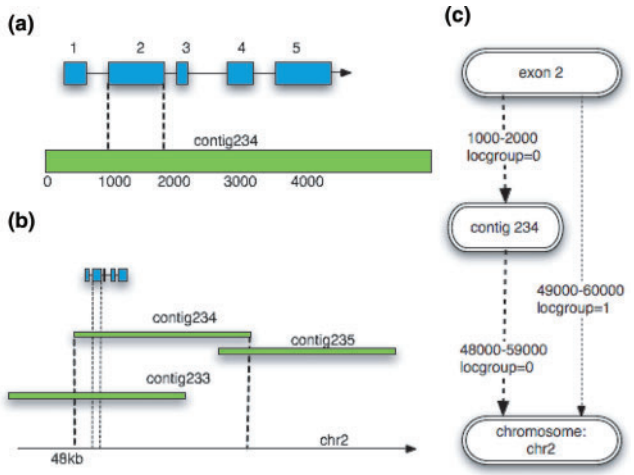

Fig. 2. Localization of features as represented in Chado. (**a**) visualization of a gene located relative to a contig feature. The exons are numbered and depicted in blue, the contig in green. Dotted lines show the relative location of the start of exon2 as 1 kb along the contig, and the end at 2 kb. (**b**) The same gene at a decreased zoom level. The contig in caption is shown located 48 k upstream of the start of chromosome 2. The thinner dotted lines show exon 2's location projected onto chr2, 49 kb upstream of the origin. (**c**) An abstract representation of rows in the Chado database, with features denoted using oval boxes and featurelocs denoted using arrows. Exon2 is located directly with respect to contig234, contig234 is located directly relative to chr2. Exon2 is also indirectly located relative to chr2 using a featureloc with locgroup set to 1.

unaware of such technicalities. However, for the purposes of software engineering and ensuring interoperability between different Chado database instances and different applications, formal treatments such as these are an essential requirement for software specifications.

We define a featureloc graph (LG) as being a set of vertices and edges, with each feature constituting a vertex, and each **featureloc** constituting an edge going from the parent **feature_id** vertex to the **srcfeature_id** vertex. The node is labeled with column values from the **feature** table, and the edge is labeled with column values from the **featureloc** table. The LG is not allowed to contain cycles: it is a directed acyclic graph (DAG). This includes self-cycles, as no feature may be localized relative to itself.

The *roots* of the LG are the features that do not themselves have an associated localization—frequently chromosomes or chromosome arms, although LG roots may also be unassembled contigs, scaffolds or features for which sequence localization is not yet known. The *leaves* of the LG are any features that are never present as a **srcfeature_id** in any **featureloc** row, typically the bulk of features, such as exons, matches and so on. The *depth* of a particular LG g, denoted $D(g)$, is the maximum number of edges between any leaf–root pair. As has been previously noted, many Chado implementations will have LGs with a uniform depth of 1. Such LGs are said to be *simple* and the features within them are said to be *singletons*. The maximum depth of all LGs in a particular database instance **i** is denoted $LGD^{max}(\mathbf{i})$.

The schema does not constrain the maximum depth of the LG. This flexibility proves useful when applying Chado to the highly variable needs of different genome projects; however, it can lead to efficiency problems when querying the database. It can also make it more difficult to write software to interoperate with the database, as the software must take into account arbitrary LG depths.

We can solve this problem by *collapsing* the LG, i.e. a graph of arbitrary depth is flattened to a depth of 1, by projecting **featurelocs** at lower levels onto the root features. The original **featurelocs** are left unaltered in the database, and redundant *inferred* **featurelocs** between leaf and root features are added to the database. In the **featureloc** table, inferred **featurelocs** are differentiated from direct **featurelocs** using the **locgroup** column. Direct (non-inferred) localizations are indicated by the locgroup column taking value 0, and transitive localizations are indicated by this column having value >0. The benefit of being able to *collapse* the localization of sequence features in inferred locations is that it allows for concise feature location implementations of any depth, while also making it possible to represent locations in the flattened (depth $=1$) format commonly used in sequence visualization tools today.

Alignments and comparative features are typically localized using pairs of **featurelocs**. Such features include hits and high-scoring pairs (HSPs) coming from sequence search programs such as BLAST, syntenic chromosomal regions and sequence variations such as SNPs (single nucleotide polymorphisms). Such features have two **featurelocs**: one relative to the query or variation feature, and one relative to the subject (hit) or reference feature. We differentiate the two **featurelocs** using the **rank** column. A **rank** of 0 indicates a location relative to the query (as is the default for most features), and a **rank** of 1 indicates a location relative to the subject (hit) feature (Fig. 3). For multiple alignments (e.g. CLUSTALW results, see Higgins *et al*., 1994), this scheme is extended to unbounded ranks $[0 \ldots n]$, with arbitrary ordering. Alignments and variant sequences may be stored in the **residue_info** column. The CIGAR format defined by the Ensembl project (Hubbard *et al*., 2005) is used for pair-wise alignments. The **feature_id, rank** and **locgroup** triple uniquely identify a **featureloc** for any particular **feature**. This means that no feature can have more than one **featureloc** with the same **rank** and **locgroup**.

The implementation of all sequence as features, *including those with and without known sequence*, combined with the **featureloc** table—which allows any feature to be localized with respect to any other feature—provides a significant expressive advantage over other biological sequence models (see Stajich and Lapp, 2006 for an overview), such as GFF3 (Eilbeck and Lewis, 2004), GenBank, BioSQL (http://www.biosql.org), BioPerl (Stajich *et al*., 2002), etc. In Chado, a feature can be localized to the most appropriate source feature. For instance, the protein product of an intron-containing gene can be localized directly as a single continuous location upon the transcript that encodes it. In other models, the location of such a protein can only be expressed as the discontinuous set of genomic locations corresponding to the protein-coding regions of the gene's exons. Thus, the Chado location more closely reflects the actual biological relationship between transcript and protein, and simultaneously avoids having to manage discontinuous locations for some features, which are prone to ambiguity.

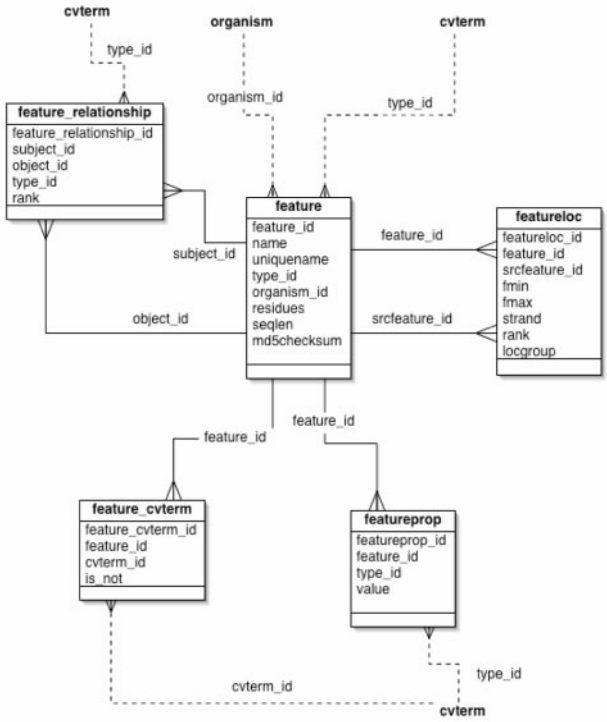

Fig. 3. This shows the main tables in the Chado sequence module. Some tables and columns have been omitted to make the diagram more concise.

4.2.3 Relationships between features Besides relating sequence features by their localization with respect to one another, it is also desirable to capture relationships that are not location based. For example, transcripts need to be related to the gene feature they are part of, likewise exon features need to be related to the transcripts they are part of, and a particular polypeptide may stand in a **derives_from** relationship to a particular mRNA molecule. The collection of such relationship links is represented in Chado using the **feature_relationship** table, and is known as the feature graph (FG). 'Subject' and 'Object' describes the linguistic role the two features play in a sentence describing the **feature_relationship**. In English, many sentences follow a subject, predicate, object word order. To say 'exons are part_of transcripts' is the correct way to describe a typical biological relationship. To say 'transcripts are part_of exons' is either grammatically or biologically incorrect.

Since the types of feature relations may be as varied as the features, ontologies are used for typing these relations in **feature_relationship**. In the case of sequence features, **feature_relationship** types are provided by either SO (Eilbeck *et al*., 2005) or the OBO relationship ontology, OBO-REL (Smith *et al*., 2005).

Some example feature graphs (FGs) are shown (Fig. 4). The FG is independent of the LG, and in general the FG and the LG should have no edges in common. If there is

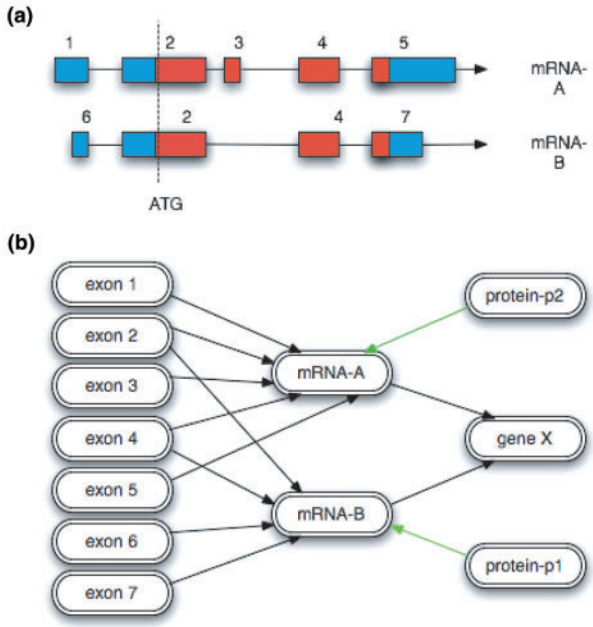

Fig. 4. A Chado central dogma feature graph for a protein-coding gene with two splice forms. (**a**) Glyph-based visualization. Boxes show exons, connecting lines show introns. UTRs shown in blue, CDS regions in red. (**b**) Feature graph depiction of rows in a Chado database: oval boxes denote features, and lines denote feature_relationships. Features are of type exon, mRNA, gene and protein. Feature_relationships are of type part_of, and green lines between protein and mRNA represent derives_from relations. Exons 2 and 4 are shared between both splice forms. Note that the feature graph conveys topological and temporal information, not, for example, linear order of exons (i.e. the exon numbers do not denote order).

a **featureloc** connecting two features, then the addition of a **feature_relationship** between these same two features is redundant. The FG is required to query the database for things such as alternately spliced genes, exons shared between transcripts, etc.

Although the Chado schema admits any FG, certain configurations are biologically meaningless, and should not be used. Unlike the LG, the FG may be cyclic, although cycles in the FG are not common. The subset of the FG corresponding to certain kinds of relationship may be acyclic, for example, the subset of the FG connecting parts with wholes via **part_of** must be acyclic.

4.2.4 Extensible feature properties The **feature** table has a very limited set of columns for recording feature attributes or properties. It was deliberately decided not to add columns such as 'anticodon' to the **feature** table, since, considering the many different types of features being stored, and the many attributes which may or may not be associated with each of those feature types, the number of columns in the table was liable to become very large and difficult to manage, with many columns being null for most features (for example, 'anticodon' does not apply to non-tRNA features).

Chado uses a single table named **featureprop** to store attributes or properties of any given feature. For sequence features, the sequence feature property ontology (see the sequence_attribute branch at http://www.sequenceontology.org/miSO/index.html) is typically used for setting property types, similarly to how it is used to handle **feature_relationship** types. This use of a controlled vocabulary instead of explicit table columns allows new properties to be added easily and without any disruption to the schema or any software that uses the schema. Provenance of each attribute is attached through the **featureprop_pub linking** table, which may identify the person who curated this feature, the sequence analysis program that predicted it or the publication in which it was first described.

4.2.5 Feature synonyms Features can have multiple names and synonyms. This is modeled in Chado with the **synonym** table, which links to features via the **feature_synonym** linking table. All **feature_synonym** links have an **is_current** Boolean attribute, which distinguishes between names in current usage and alternate or obsolete names. Depending on the experimental history of a feature, multiple features can potentially share a common synonym, and a single feature can have multiple synonyms. The provenance of a particular synonym is indicated using the **pub_id** foreign key, which references the **pub** table.

4.2.6 Feature annotations Detailed annotations of features, by associating features with types from the gene ontology (GO, Ashburner *et al.*, 2000) or the cell ontology (Bard *et al.*, 2005), can be accomplished using the **feature_cvterm** linking table. Multiple ontology terms may be associated with each feature, and terms from multiple ontologies may also be associated with each feature. Provenance data can be attached with the **feature_cvtermprop** and **feature_cvterm_dbxref** higher-order linking tables.

It is up to the curation policy of each individual Chado database instance to decide which kinds of features will be linked to ontology and other controlled vocabulary terms using feature_cvterm. Some may link terms to gene features, others to the distinct gene products (processed RNAs and polypeptides) linked to the gene features.

4.3 The Chado SQL API

Chado has a library of SQL functions for performing useful operations on biological data. The function library is organized by the corresponding parent schema module. Some functions include:

- Range operations on features and featurelocs; for example
  - Intersection
  - Overlap
  - Containment
  - Range difference
- Projection (not yet implemented)
- Operations on biological sequences; for example
  - DNA reverse complementation
  - DNA to amino acid translation

  - Splicing features together, such as exons in a transcript
- Graph operations on ontologies; for example, dynamic transitive closure of relations
- Operations on phylogenetic trees; for example
  - Calculating height and depth of any node in a tree
  - Testing for monophyleticity
- Curation dataflow operations; for example
  - Providing new **dbxref** identifiers using standard rules
  - Splitting and merging of features

All of these functions could be encoded in application software or the middleware, and in many cases they will be. However, there is tremendous benefit to also providing these functions within the DBMS, since they can then be incorporated into SQL queries, increasing the power and efficiency of the query language.

Each function has an *interface definition*, which is DBMS independent, and DBMS specific *implementations*. Currently, the only implementations are for the PostgreSQL specific PL/PgSQL language. However, most could be converted to standard SQL99 syntax fairly easily, and thus implemented in other DBMSs such as DB2 and Oracle.

In the PostgreSQL implementations, we have used DBMS specific data types to extend standard SQL, in order to provide faster operations on such things as range operations (http://www.iscb.org/ismb2003/posters/hlappATgnf.org_326.html) These implementation details are hidden from the software that calls the functions, except in so far as to make them function faster.

4.4 Software interoperation and bridge layers

Chado sequence features have been incorporated into current versions of a number of software applications, for example: Apollo (Lewis *et al*., 2002), Gbrowse (Stein *et al*., 2002) and CGL (Yandell *et al*., 2006). As previously noted, Chado's use of the **feature** table to store all features, no matter what the type, is useful for reducing schema changes. However, it can have the detrimental effect of introducing an extra layer of abstraction, complicating simple queries. For examples, a query to determine how many genes there are in the database requires a join and constraint on the name column in the **cvterm** table. The views, or bridge layers, are our solution to this problem.

4.4.1 DBMS views All major DBMSs allow the defining of database views; a view is a kind of virtual table, defined in terms of tables or other views. Views are used to present a simplified means of querying the database; this is particularly useful for helping users[3] to form queries. For the purposes of querying the database, views act just like tables; the DBMS will transparently rewrite any query that references a view into one using the corresponding tables. Views can also be used to write into the database, but this requires the coding of trigger functions particular to each view.

Views can also be used as an insulation layer, buffering applications from changes in the schema. Schema evolution has the potential to incur high software development costs, as changes in the database can percolate into multiple applications or middleware code. If the schema is modified (in a non-backwards compatible way), views can in effect emulate previous versions of the schema.

Views can also be used to define a common 'export schema' to allow applications to interoperate with multiple different schemas. This can also be done to support application development, in cases where a simplified version of a complex schema can be presented via a view layer to certain applications. In a similar vein, views can be used to make one schema appear to be another schema, we call this a *bridge* or *compatibility* layer.

The collective name for both tables and views is a *relation*.[4] Chado is a relational schema, and as such is a collection of definitions of relations. These definitions are supplied as table definitions as a matter of convenience, but there is no reason why any of these tables should not be swapped out for an equivalent view (with the provision that the appropriate trigger functions are created if the database is to be updated). This may occasionally be desirable for performance reasons.

Views can also be turned into tables, which can then be indexed for efficiency. This is known as view *materialization*. Materialized views are most useful in situations where data is read frequently and modified rarely or never, such as a report/warehouse instance of a database.

Views are indispensable for both querying and developing interoperable software for complex relational databases such as Chado, where the high degree of abstraction and normalization makes this more difficult. In Chado, the views are organized by module. Name clashes between core Chado tables and views are avoided by placing the views in different *tablespaces* (also referred as *schemas* in the context of the PostgreSQL DBMS).

4.4.2 Chado bridge layers SOFA is a stable subset of the entire SO; releases are made on a yearly basis. The SOFA layer provides a view (or table) for every type in the SOFA ontology. This stability means that software coded to the SOFA layer will be less prone to the negative effects of schema evolution. The SOFA layer is generated automatically and a default layer is provided. Chado database administrators can choose to regenerate this layer as a materialized view (table) layer if desired.

Bridge layers are also available for the GO database (Harris *et al*., 2004), which is a resource made available by the GO Consortium, and a core part of the GO DB is a set of tables for modeling ontologies, with many similarities to the Chado CV module. These similarities make it relatively straightforward to define a collection of views over a Chado database which allow it to function with software developed for the GO Database, including software developed by the GO Consortium such as

[3]In this context, 'users' refers to advanced users interested in data mining, as well as programmers writing report software, as opposed to bench biologist end-users.

[4]Not to be confused with the sense of how this word is used in ontologies.

the AmiGO ontology browser (see http://www.godatabase.org/cgi-bin/amigo/go.cgi).

4.5 Reuse and the GMOD project

The task of effectively managing genome-scale biological data is an enormous software engineering challenge. Given finite resources, it would appear wise to share software components as much as possible. This was the driving force behind the creation of the GMOD project. Supported by several of the major model organism projects, who have contributed funding, software and developers, the GMOD project has become a major organizational force in the development of software by model organism projects around the world. GMOD required that its adopted schema be *adaptable*, in order to be quickly altered when new types of data or relations are demanded to faithfully represent the science; that it have a suitably *flexible* model, that can represent only those data that are germane to a particular research area; that it must mirror the precision of the experimental data, and therefore the representations should support a varied *range of classification precision*; that it must offer effective and reliable *query and inference support*; that it must be *interoperable* with other software; and finally that it must *reduce maintenance costs*. GMOD now employs Chado as its common database schema.

5 DISCUSSION

Chado is a highly generic, flexible schema. This allows a uniform schema to be used by different projects covering a variety of genomes, without having to anticipate in minute detail the individual requirements of each of these projects. It also provides a large amount of future-proofing of the schema in the face of changes in both our understanding of biology and changes in the experiments and processes related to biological discovery. Without this flexibility, the schema would be in a constant state of migration, incurring large costs on developers and users. In addition, the schema would most likely bifurcate whenever different software developers have cause to modify it to suit their individual projects' needs.

We note that such flexibility in a schema is not without its own costs. The most significant hurdle in adopting Chado is establishing best practices for implementation in a schema that is so flexible. Without constraints, different groups are liable to model similar kinds of data in different ways. For example, in FG typing and topology there are many choices to make when representing a gene model: should intron and UTR features be manifested (i.e. exist as actual rows in the **feature** table) or left implicit (i.e. be derived by software as needed)? Which SO type should be used to represent the translational product of an mRNA: CDS or polypeptide? In both cases the choices are logically valid, but software that is not aware of the practical equivalence of these two types will not work equally well across these different schema instantiations. Without clear understanding and guidelines for implementing data in Chado using current ontologies, it will be difficult, if not impossible, to continue to develop interoperable application software of any kind of complexity. Whilst the use of ontology within the schema is of great benefit, there is still a significant task facing application developers, who must grasp the fundamentals of the structure and logic of ontologies if they are to achieve interoperability.

Chado's flexible schema is less generic than the RDF model (Brickley and Guha, 2000), although the two are comparable as can be seen in the 'subject–predicate-object' triple pattern employed in Chado. However, there are also differences between the systems, as can be seen by the fact that Chado triples often take additional arguments (such as, for example, in the case of ordering exons in a spliceform) in contrast to the strict 3-ary pattern in RDF. However, it is our intention to be compatible and reuse technology as appropriate so we continue to pay attention to the evolution of RDF and attendant technology.

Future plans for Chado include the creation of new modules and refinement of existing modules for other domains, such as genotype–environment–phenotype associations. This will also require the development of ontologies for these domains. We also intend to do more work a benchmarking suite that will allow us to target areas in which we can make the query response times faster.

In summary, Chado uses a novel method of constraining the variety of representations whilst retaining useful flexibility, through the use of ontologies and layered compliance levels. Chado has successfully answered our initial question, whether we could create a relational database schema that could and would be reused, as attested by the number of groups that have adopted Chado for production systems. Besides FlyBase, these groups include: The Institute for Genomic Research; BeetleBase (Wang *et al.*, 2007); SpBase at the California Institute of Technology for Sea Urchins; DictyBase at Northwestern University; GeneDB at the Sanger Institute for pathogens (e.g. *Trypanosoma brucei*, *Leishmania major*, *Plasmodium falciparum*, *Staphylococcus aureus*, *Salmonella typhi*, *Schizosaccharomyces pombe* and scores more); Sol Genomics Network at Cornell University for *Solanaceae* (tomatoes, etc.); University of Utah, Sanchez Laboratory, for *Schmidtea mediterranea* (planaria); *Xenopus* genome database at the University of Calgary; the Paramecium database at the Centre National de la Recherche Scientifique, (Arnaiz *et al.*, 2007); Princeton University MicroArray database and the Saccharomyces Genome Database-Lite; the National Evolutionary Synthesis Center for *Heliconius* (butterfly); the Daphnia Water Flea Genome Database Indiana University; VectorBase for A. gambiae at Notre Dame University; the University of Wisconsin at the Wicell Research Institute; Infobiogen in France; and at the University of California Los Angeles both the Patricia Johnson Laboratory, for studies of *Trichomonas vaginalis* and the Stan Nelson Laboratory for studies in human genetics using microarrays. While our technical solution is not a panacea for the challenge of accurately reflecting biological knowledge in a computable format, it is a step in the right direction. By leveraging the representation of biology encoded in ontologies, the physical representation (in the schema) is more adaptable and flexible, and thus reduces maintenance costs; at the same time the ontologies can be used to support varied precision in classification, more effective query and inference support, and support interoperability with other software.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the support and advice from all of our colleagues in the FlyBase Consortium, whose expertise and initiative led to Chado's development. We also gratefully acknowledge the input from everyone in the GMOD Consortium, especially Scott Cain, Brian Osborne and Guanming Wu for carrying this work forward for implementation by other database groups. FlyBase is supported by a grant from the Public Health Services (NIH grant 5P41 HG000739, through the National Human Genome Research Institute, W. Gelbart, PI) with additional support for Chado development from the HHMI (G. Rubin, PI). Finally, we are extremely appreciative to Chado's users whose feedback and support continues to improve Chado and its associated software.

FlyBase consortium contributions Current and former FlyBase Consortium members making notable contributions to this project are William M. Gelbart, Aubrey de Grey, Stan Letovsky, Suzanna E. Lewis, Gerald M. Rubin, ShengQiang Shu, Colin Wiel, Peili Zhang and Pinglei Zhou.

Conflict of Interest: none declared.

REFERENCES

Arnaiz,O *et al.* (2007) ParameciumDB: a community resource that integrates the *Paramecium tetraurelia* genome sequence with genetic data. *Nucleic Acids Res.*, **35**, D439–D444Epub.

Ashburner,M *et al.* (2000) Gene ontology: tool for the unification of biology. The gene ontology consortium. *Nat. Genet.*, **25**, 25–29.

Bard,J *et al.* (2005) An ontology for cell types. *Genome Biol.*, **6**, R21.

Beadle,G.W. and Tatum,E.L. (1941) Genetic control of biochemical reactions in neurospora. *Proc. Natl Acad. Sci.*, **27**, 499–506.

Brickley,D. and Guha,RV. (2000) Resource description framework (RDF) schema specification 1.0, *W3C Candidate Recommendation*.

Clark,T. *et al.* (2004) Globally distributed object identification for biological knowledgebases. *Brief Bioinform.*, **5**, 59–70.

Durbin,R. and Theirry-Mieg,J. (1994) ACeDB. *Computational Methods in Genome Research*. Plenum, New York.

Eilbeck,K. and Lewis,S. (2004) Sequence ontology annotation guide. *Comp. Funct. Genomics*, **5**, 642–647.

Eilbeck,K. *et al.* (2005) The sequence ontology: a tool for the unification of genome annotations. *Genome Biol.*, **6**, R44.

Harris,MA. *et al.* (2004) The Gene Ontology (GO) database and informatics resource. *Nucleic Acids Res.*, **32**, D258–D261.

Higgins,D. *et al.* (1994) CLUSTALW: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.*, **22**, 4673–4680.

Hoskins,RA. *et al.* (2002) Heterochromatic sequences in a Drosophila whole-genome shotgun assembly. *Genome Biol.*, **3**, RESEARCH0085.

Hu,J. *et al.* (2001) The ARKdb: genome databases for farmed and other animals. *Nucleic Acids Res.*, **29**, 106–110.

Hubbard,T. *et al.* (2005) Ensembl. *Nucleic Acids Res.*, **33**, D447–D453.

Lewis,SE. *et al.* (2002) Apollo: a sequence annotation editor. *Genome Biol.*, **3**, RESEARCH0082.

Morgan,TH. (1907) The cause of gynandromorphism in insects. *Am. Nat.*, **41**, 715–718.

Mungall,CJ. *et al.* (2002) An integrated computational pipeline and database to support whole-genome sequence annotation. *Genome Biol.*, **3**RESEARCH0081.

Smith,B. *et al.* (2005) Relations in biomedical ontologies. *Genome Biol.*, **6**, R46.

Stajich,JE. *et al.* (2002) The Bioperl toolkit: Perl modules for the life sciences. *Genome Res.*, **12**, 1611–1618.

Stajich,JE. and Lapp,H. (2006) Open source tools and toolkits for bioinformatics: significance, and where are we? *Brief Bioinform.*, **7**, 287–296.

Stein,LD. *et al.* (2002) The generic genome browser: a building block for a model organism system database. *Genome Res.*, **12**, 1599–1610.

Wang,L. *et al.* (2007) BeetleBase: the model organism database for *Tribolium castaneum*. *Nucleic Acids Res.*, **35**, D476–D479.

Yandell,M. *et al.* (2006) Large-scale trends in the evolution of gene structures within 11 animal genomes. *PLoS Comput. Biol.*, **2**.