A Tool for Collaborative Construction of Large Biological Ontologies

Jie Bao[a], Zhiliang Hu[b], Doina Caragea[a], James Reecy[b], Vasant G Honavar[a]
[a]Artificial Intelligence Research Laboratory, Department of Computer Science
[a]Center for Computational Intelligence, Learning, and Discovery
[b]Department of Animal Science
Iowa State University, Ames, IA 50011, USA
Email: {baojie, zhu, dcaragea, jreecy, honavar}@iastate.edu

***Abstract*— Ontologies that identify and define entities and relationships in specific domains of interest offer a powerful approach for annotating biological data in a form that allows users and software tools to retrieve, interrelate, and extract biological knowledge. In order for such ontologies to be broadly useful to the scientific community, they need to capture knowledge and expertise of multiple experts and research groups. Consequently, the construction of such ontologies necessarily requires collaboration among individual experts or research groups. Such collaboration may be direct (through collaborative creation of an ontology) or indirect (through reuse or adaptation of previously published, autonomously developed ontologies). Support for such collaboration is largely lacking in existing ontology development environments. We describe some initial steps towards the development of a collaborative ontology development environment. Specifically, we describe an ontology editing tool INDUS DAG Editor which exploits the notion of modular ontologies (or ontology packages) to support sharing, reuse, and collaborative editing of partial order (i.e., DAG-structured) ontologies. INDUS DAG Editor can engage diverse and relatively autonomous communities of biologists in the process of creating the ontologies needed for annotating, integrating, and analyzing diverse sources of 'omics' data.**

I. Introduction

The transformation of biology from a data-poor science into increasingly data-rich science has led to a proliferation of repositories of biological data. For example, Discala et.al. [4] reports that there are in excess of 500 databases of interest to molecular biologists alone. In order for sharing, integration, and use of such data among individuals and research groups to be possible, it is necessary for each data source to be annotated using controlled vocabularies or more generally ontologies that correspond to conceptualizations of objects, attributes, and relationships among attributes, within the respective domains of interest. Consequently, there are many efforts directed at the development of such ontologies e.g., Gene Ontology (GO)[1], Plant Ontology (PO)[2] and Phenotype and Trait Ontology (PATO)[3].

Ontologies that are intended to be useful to specific scientific communities often consist of thousands of terms. For example, the Gene Ontology contains 2×10^5 terms and the Gramineae Taxonomy contains 7×10^5 terms. Furthermore, such ontologies have to capture the collective knowledge and expertise of multiple experts and research groups. An unavoidable consequence of the increasing size and complexity of ontologies is the need for *collaboration* among multiple experts or research groups. Such collaboration can be either direct (as in collaborative creation of an ontology), or indirect (through the reuse of previously published, autonomously developed ontologies). In such a setting, a large ontology is built and curated by a community, with each of its members contributing only a small part of the ontology. For example, the GO consortium involves more than a dozen research groups and hundreds of contributors. With the rapid proliferation of biological ontologies, there is an increasing need for tools that enable reuse of fragments of independently developed ontologies to assemble domain, context, user or community-specific customized ontologies.

Unfortunately, existing ontology editing tools such as the DAG-Edit [3] and OBO-Edit [8] offer extremely limited support for such collaborative development. Consequently, there is an urgent need for tools for creating and processing increasingly large, collaboratively developed ontologies, with limited time and space resources. To address this need, we have developed extensions to ontology languages and software tools for *collaborative ontology building* (COB). Our approach builds on recent advances in modular ontologies [1] to facilitate the creation and management of large biological ontologies within a distributed curator model.

This paper is organized as follows: Section II discusses the requirements for COB environments (using Animal Trait Ontology as a concrete example), and the limitations of current CVS-based approaches to COB. Section III introduces our *ontology package* approach and a software tool for collaborative construction of biological ontologies. Section IV discusses related research. Section V presents conclusions and future research.

II. Desiderata of COB in Biology

The process of constructing a small-scale ontology is typically *non-collaborative*, which means that it involves only a single curator (see Fig. 1). Such an ontology is usually stored in one or several files. When editing the ontology with an available tool (e.g., Protege, DAG-Edit), the curator needs to make a local copy of this ontology. After editing, the initial ontology is completely replaced with the new version that is

[1]http://www.geneontology.org/
[2]http://www.plantontology.org/
[3]http://obo.sourceforge.net/cgi-bin/detail.cgi?poav

the result of the editing process.

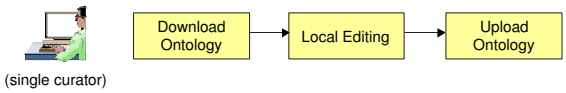

Fig. 1. Non-collaborative Ontology Building

In contrast, as noted earlier, the process of constructing large-scale biological ontologies is necessarily *collaborative*. Animal Trait Ontology (ATO), an ontology being developed to serve as a controlled vocabulary for description of animal traits (i.e., phenotypes, such as *meat quality*, *health*, *external appearance*, *reproduction*, etc.) at the desired level of detail. ATO is intended to support: meaningful crossboard trait comparisons in genetics and genomics studies; annotating the genome; and cross-species comparisons. We draw on our experience with the construction of ATO to articulate some of the requirements of a COB environment.

A. Requirement of COB Environments

A COB environment needs to offer support for several tasks:

Knowledge Integration: The target ontology typically requires integration of ontology fragments contributed by multiple participants. For example, ATO needs to integrate contributions from individuals or research groups with expertise with regard to specific species.

Concurrence Management: Different curators need to be able to work on different parts of the ontology simultaneously. Suppose that curator A downloads the current version of ATO and performs local editing on “PigMeatQuality”; before A submits the modified version of ATO, another curator B may submit changes on “PigHealth”. Therefore, COB environment must ensure that when A submits his or her locally edited copy, the version updated by B is not inadvertantly overwritten.

Consistency Maintenance: Components of the same ontology developed by different curators may be inconsistent since an ontology usually reflects the local point of view of each curator. For example, “PigLoinWeight” may be taken as a sub-trait of the “PigMeatQuality” trait by one expert, and as a “PigProduction” trait by another expert. Inconsistencies can arise as a result of some previously defined ontology terms being rendered obsolete. For instance, while a user A is in the midst of defining a term “NumberOfTeatsOfSow” under “ExteriorTraits”, another user B may be eliminating “ExteriorTraits” (e.g., by merging it with with ”ReproductiveTraits”) and submiting the change before user A does. In both cases, ensuring the semantic consistency of the resulting ontology requires reconciliation of different points of view.

Privilege Management: In order to ensure the accuracy of the ontology, the COB curator community needs to include individuals with different levels of privileges, based on their expertise, authority, and responsibility. For instance, a curator A may be responsible for all pig traits (all terms under “SusScrofa”), while a curator B may be responsible only for pig health traits (all terms under “PigHealth”).

History Maintenance: COB environments should have mechanisms to recover from wrong, unintended or even vicious changes to an ontology. Therefore, changes to an ontology must be recorded in order to be able to track the authorship of a change and to prevent loss of important information.

Scalability: Many ontologies consist of tens of thousands of terms. Consequently, the COB process has to be scalable to large ontologies. If curator A only intends to edit the ontology component about pig meat quality, it is neither necessary, nor desirable to make A download and submit the parts of ATO that are not affected by his or her work. Biological ontologies are often *partial order ontologies*. Such ontologies can be represented as directed acyclic graphs (DAG). Although general purpose ontology editing tools (e.g., Protege) based on rather expressive ontology languages (e.g., OWL) can be used for DAG-structured ontologies, they are much less efficient than ontology editing tools that specialize in DAG-structured ontologies (e.g., DAG-Edit). COB tools can be used to exploit the DAG structure of biological ontologies to ensure scalability.

B. CVS-based Collaboration and its Limitations

There is a growing awareness in the biological ontology community about the need for collaboration in the construction of large biological ontologies. Consequently, there is a growing interest in the development of COB environments and tools. The Gene Ontology consortium represents one of the most successful collaborative efforts aimed at creating biological ontologies. The GO collaborative building process [2] is based on a concurrent versions system (CVS), a request tracking system hosted on SourceForge, and natural language communications among GO users and curators (facilitated by several email lists). This process consists of the following major steps (Fig. 2):

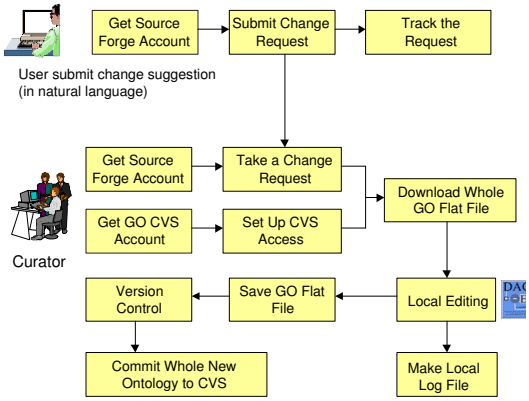

Fig. 2. Collaborative Ontology Building with CVS: Gene Ontology

1) A user submits a change request on SourceForge.
2) One of the GO curators claims the request.
3) The curator downloads the GO flat file from CVS.
4) (optional) The curator declares “ownership” of the GO terms in a file (GO_numbers) in the CVS.
5) (optional) The curator sends emails to other curators to avoid conflicting.

6) The curator edits the flat file downloaded (e.g., with DAG-edit) and saves the modified file.
7) The curator compares the local flat file with the current version in the CVS repository and merges all changes made by other curators after his or her last download.
8) The curator uploads the modified ontology to CVS.

CVS and email list based collaboration approach, while useful, fail to satisfactorily address the key requirements of COB environments enumerated above because of the following drawbacks:

Unprincipled Authorization and Organization: There is no principled mechanism to ensure curator privilege assignments, nor clear organizational division of the whole ontology into smaller manageable units.

Risk of Inconsistency: There is no principled way to avoid unintended couplings and over-writing. The validity and consistency of the ontology are dependent almost entirely on the discipline exercised by the human curators (e.g., the habit of checking differences between versions before submission) and good community communications (e.g., via email lists). Hence it is not surprising that there are many inconsistencies in GO [10].

Lack of Support for Editing or Reuse of Parts of the Ontology: A curator has to download the *entire* ontology, before editing, and submit the *entire* modified ontology, after editing, although a small part of the ontology may actually be affected by the changes. Similarly, a user cannot download and reuse only a selected subset of GO, e.g., the part of GO that is concerned with description of Kinases.

Expensive History Maintenance: In CVS version control, even a minor edit of the ontology, e.g., editing of a single term, relationship or property, causes the ontology file to be replicated in its entirety. In addition, tracing the changing history of a term requires processing the entire ontology text file for comparisons. As a consequence, retrieving the relevant information about editing history of a relevant ontology fragment is rather expensive.

Limited Participation: In the absence of a principled way to grant different levels of privileges to different types of users (e.g., core curators versus normal curators), and a handy tool to accept/deny/modify/revert *local* changes made by other curators, the curator community has to be limited to a small number of trusted curators. Contributions from outside the trusted curator community can only be offered in natural language, via the request tracking system or email lists. This limits the participation of the broader scientific community in the ontology building process. The small number of trusted curators who can respond to input from the community results in a bottleneck that slows down the rate at which ontology can change in response to community input.

Therefore, success of the CVS-based approach to COB relies heavily on the implicit community commitment and cooperation, and on the self-discipline of the involved participant as opposed to mechanisms that are designed specifically to support collaboration. Hence, there is an urgent need for knowledge representation formalisms, as well as systems and and software for COB. In what follows, we describe an approach based on an organization of a complex ontology into ontology modules or packages [1].

III. COB: An Approach Based on Modular Ontology Packages

Many of the drawbacks of current approaches to COB arise from lack of support for localizing the interactions among different parts of a large ontology. The primary source of this difficulty is the lack of an organizational structure which forces us to treat an ontology in its entirety. In other words, the current state of knowledge representation languages used for specifying ontologies is reminiscent of the early programming languages which lacked support for organizing programs into coherent units (e.g., subroutines).

A. Organizing Ontologies into Packages

We observe that an ontology that results from collaboration among multiple experts or research groups can be viewed as consisting of smaller modules, called **packages** [1]. Each such package encapsulates a closely related set of terms and relations between terms. Together, these terms and relations represent the ontological commitments of an individual expert or a research group (or a sub-community) regarding a small, coherent part of the universe of discourse (e.g., traits of interest to the livestock community).

Definition 1 (Package): Let S be the set of all terms and R be the set of all relations of an ontology $O = (S, R)$. A package $P = (S_P, R_P)$ of the ontology O is a fragment of O, such that $S_P \subseteq S$ and $R_P \subseteq R$. A term or relation $t \in S_P \cup R_P$ is said to be a *member* of P, i.e., $t \in P$. P is called the *home package* of t.

For example, ATO is an ontology, Pig is a package in ATO, "LoinEyeArea" and "MeatQuality" are terms in Pig; *subclassOf* ("LoinEyeArea", "MeatQuality") is a relation in Pig; "LoinEyeArea" is a member of Pig and Pig is the home package of "LoinEyeArea".

Note that the relations in an ontology package specify its *semantic structure*. However, COB calls for recognition of the *organizational structure* of an ontology.

The package-based ontology language [1] offers language constructs that allow an ontology package to be declared as a *sub package* of another package. Such *organizational* relations among packages allow specification of the organizational structure of an ontology in terms of hierarchical nesting of packages.

Definition 2 (Package Nesting): A package P_1 can be nested in one and only one other package P_2. This is denoted by $P_1 \in_N P_2$. P_1 is said to be a *sub package* of P_2 and P_2 is the *super package* of P_1. The collection of all package nesting relations in an ontology constitutes the *organizational hierarchy* of the ontology.

For example, the package Pig can contain smaller packages, such as PigMeatQuality and PigHealth. PigMeatQuality (the sub package) is nested in Pig (the super package).

Modular structure of ontologies can be exploited in COB in several ways:

- **Division of Labor**: A package consists of a set of closely related terms and relations in a smaller sub domain (e.g., pig traits) of a main domain. Therefore, a package can be assigned to curators or experts with the best knowledge of the relevant sub-domain. The package hierarchy helps organize and manage interactions among collaborating groups of sub-domain experts with different degrees and scopes of expertise.
- **Scalability**: When the ontology has a modular structure, a curator is not required to work with the entire ontology. Instead, the curator can download and edit one or more packages that fall within his or her (sub) domain of expertise, while other packages are being edited by other curators. Because the effects of edits are *localized*, different curators can independently work on packages e.g., PigMeatQuality and PigHealth, that belong to different parts of the organizational hierarchy. This simplifies the task of propagating the effect of changes within individual packages through the rest of the ontology. The result is significant reduction in communication overhead, computational cost (e.g., parsing, consistency check), memory requirements, as well as the cost of history tracking in collaborative editing of very large ontologies.
- **Partial Reuse**: Ontologies with modular structure can be partially reused. For example, a user interested in pig traits would only need to download packages about pig (e.g., Pig and PigMeatQuality), and avoid having to deal with ontology packages such as Cattle and Horse which are no interest to the user.
- **Broadened Participation**: One of the reasons behind the success of the world-wide web is the network effect: A large number of users were able to contribute modules (web pages) that make up the world-wide web. The power of the network effect is confirmed by experience with DMOZ and Wikipedia projects. Effective COB environments can enable broader participation of the scientific community in ontology building efforts without sacrificing the quality of the resulting ontology. Specifically, a user with the appropriate privileges can make a change to the PigMeatQuality package (as opposed to just proposing such a change for consideration via an email list); this change could be approved or denied by a curator with higher privileges.

B. The INDUS DAG Editor

Based on our package-based ontology framework, we have developed ATO Editor[4], a modular ontology editor for building and deploying large animal trait ontologies. ATO Editor is a customized version of INDUS DAG Editor, a collaborative ontology building environment for partial order ontologies. The editor allows ontology developers to create a community-shared ontology server with remote database storage and support for concurrent browsing and editing of an ontology. Specifically, the current implementation of the INDUS DAG editor offers support for:

- **Modular Ontologies**: A large ontology is organized into packages with nested scope. This helps localize the effects of changes to an ontology in a setting where multiple experts collaborate in developing a large ontology.
- **Database Storage for Ontologies**: The ontology is stored on a relational database server (e.g., Postgres); a user can connect to the server and check out one or more packages, edit them, and check them back in the database (when finished with editing). Database storage allows retrieval of only the relevant parts of an ontology (as opposed to the entire ontology).
- **Concurrent Editing**: Multiple users can concurrently edit an ontology, through appropriate locking mechanisms, without a user inadvertently overwriting the work of others. Thus, different modules of an ontology can be developed by different authors. A user can edit an ontology by editing one or more packages. The packages being edited are locked for writing and unlocked when a the user submits the edited version(s) of the checked out package(s)
- **Change Tracking**: When a user submits an updated version of an ontology module, the ontology server automatically creates the change log for affected terms and relations in the package. This feature can be used to track the history and if necessary, to revert a term or a relation to its previous version.
- **User Privilege Management**: Authors of the ontology can have different levels of privileges (such as ontology administrator, and package administrator) over modules in an ontology. The author of a package can authorize other users the access to certain terms, therefore controlling the extensibility of that package.
- **Graphical User Interface**: Users can browse and edit DAG-structured ontologies using a graphical user interface (GUI).
- **Import and Export of Ontologies in Multiple Formats**: Users can import and export ontologies in several standard formats, such as OBO (Open Biomedical Ontologies) and OWL into and out of the INDUS DAG Editor.

Furthermore, using INDUS DAG Editor, users can create an ontology that consists of both *is-a* as well as *part-of* hierarchies, with some terms being shared by both hierarchies (if necessary). Such ontologies are quite common in life sciences.

The editing process in INDUS DAG Editor is summarized in Fig. 3.

The editor is written in Java to ensure its universal portability across different hardware and operating system platforms.

IV. Related Work

With the growing need for ontologies, several research groups have tried to develop ontology editors [6]. However, most of the existing ontology editors, including the most widely used ontology editors, such as Protege[5], OilEd[6] and

[4]http://www.animalgenome.org/bioinfo/projects/ATO/

[5]http://protege.stanford.edu/

[6]http://oiled.man.ac.uk/

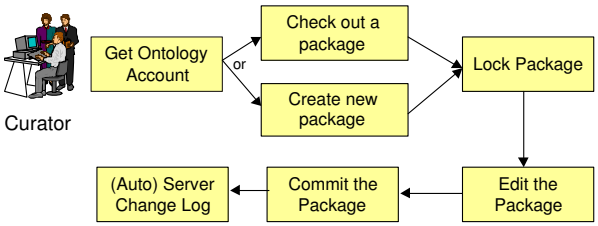

Fig. 3. Collaborative Ontology Building with Package-extended Ontology

DAG-Edit [3], provide at best extremely limited support for *collaborative* ontology development. Support for collaborative ontology development has begun to receive some attention, for example in systems such as CODE [7], OntoEdit [7], Ontolingua [5], and WebODE [9]. Most of them provide concurrent access control with transaction oriented locking, and in some cases, even rollback. However, to the best of our knowledge, none of the existing ontology editors offer support for a modular ontology representation - a feature that is critical for collaborative development of large ontologies by communities of users. Many of the current ontology editors use file-based centralized storage - that is, the entire ontology is stored in a large file. Consequently, parsing, editing, consistency checking, and change tracking processes do not scale to large ontologies. Even the systems that provide database storage for ontologies (e.g., Protege-OWL) use in-memory models of the whole ontology for editing and reasoning. In contrast, the COB tools proposed in this paper complement existing ontology development tools in areas where there is a need for active collaboration among experts in developing ontologies in domains where the complete ontology can be naturally viewed as consisting of many loosely coupled modules.

V. Summary and Discussion

Ontologies that identify and define the entities and relationships in specific domains of interest offer a powerful approach for annotating biological data in a form that allows users and software tools to retrieve, inter-relate, and extract biological knowledge from such data. With the rapid proliferation of data sources, services, and ontologies, there is an urgent need for software tools for collaborative development of ontologies, as well as sharing and integration of independently developed ontology fragments.

Although there are several existing tools for developing ontologies, there are few that support collaborative development of large biological ontologies. The INDUS DAG Editor aims to provide a tool for active collaboration among a community of experts in developing a large ontology. The ATO Editor, a version of the INDUS DAG Editor, designed for use by the Animal Genomics Community, offers a proof of concept of a software tool based on our approach to development of language constructs that allow specification of an organizational structure of a complex ontology into a hierarchy of modules, system features (database storage, use privilege management, change tracking, etc).

Work in progress is aimed at:

1) Promoting collaborative development and use of ATO by engaging a community of users with expertise in the relevant (sub) domains
2) Further development of COB tools to allow collaborative development of other ontologies (e.g., OBO - Open Biomedical Ontologies) including ontologies that are more expressive than partial order ontologies that are currently supported by INDUS DAG Editor
3) Accommodation of existing ontologies (such as GO) within the COB environment through appropriate partitioning of such ontologies into packages organized into a coherent organizational hierarchy
4) Inference mechanisms to ensure the consistency of an ontology package in settings more general than partial order ontologies
5) Support for semantic integration of semantically heterogeneous ontologies via inter-ontology mappings
6) Development of reasoning support for identifying the maximal consistent subset(s) independently developed ontologies

Acknowledgment This research is supported in part by grants from the NSF (IIS 0219699), the NIH (GM 066387) to Vasant Honavar and the USDA NAGRP Bioinformatics Coordination Project to James Reecy and research assistantship support for Jie Bao from the Center for Integrative Animal Genomics (CIAG), Iowa State University.

References

[1] Jie Bao, Doina Caragea, and Vasant Honavar. Towards collaborative environments for ontology construction and sharing. In *International Symposium on Collaborative Technologies and Systems (CTS 2006)*. 2006.

[2] Gene Ontology Consortium. Go editor guides. In *http://www.geneontology.org/GO.contents.curator.guides.shtml*. 2005.

[3] John Day-Richter. Dag-edit: A controlled vocabulary editor. In *http://www.godatabase.org/dev/java/dagedit/docs/*. 2004.

[4] C. Discala, X. Benigni, E. Barillot, and G. Vaysseix. Dbcat: a catalog of 500 biological databases. *Nucleic Acids Research*, 28(1):8–9, 2000.

[5] A. Farquhar, R. Fikes, W. Pratt, and J. Rice. Collaborative ontology construction for information integration. In *Technique Reports of Knowledge Systems Laboratory, Department of Computer Science, KSL-95-63*. 1995.

[6] Asun Gomez-Perez, Juergen Angele, Mariano Fernandez-Lopez, V. Christophides, Athur Stutt, and York Sure. Ontoweb deliverable 1.3: A survey on ontology tools, http://ontoweb.org/about/deliverables/d13_v1-0.zip/. 2002.

[7] P. Hayes, R. Saavedra, and T. Reichherzer. A collaboration development environment for ontologies. In *Proceedings of the Semantic Integration Workshop, Sanibel Island, Florida,*. 2003.

[8] Chris Mungall. Integrated ontologies for biological annotation: The national center for biomedical ontologies. In *The First International Biocurator Meeting. Pacific Grove, CA, December 8-11, 2005*. 2005.

[9] Julio César Arpírez Vega, Óscar Corcho, Mariano Fernández-López, and Asunción Gómez-Pérez. Webode: a scalable workbench for ontological engineering. In *K-CAP*, pages 6–13. 2001.

[10] Iwei Yeh, Peter D. Karp, Natalya Fridman Noy, and Russ B. Altman. Knowledge acquisition, consistency checking and concurrency control for gene ontology (go). *Bioinformatics*, 19(2):241–248, 2003.

[7] http://www.ontoknowledge.org/tools/ontoedit.shtml