Collaborative Package-based Ontology Building and Usage

Jie Bao and Vasant Honavar
Artificial Intelligence Research Laboratory
Computer Science Department
Iowa State University
Ames, IA USA 50010
Email: {baojie, honavar}@cs.iastate.edu

Abstract

With the increased complexity of ontologies, building and using large-scale ontologies require cooperative and scalable mechanisms. In this paper, we propose a modular ontology approach for such challenge. A large ontology has multiple smaller modules called packages, which are sets of highly related terms and term relations; packages can be nested in other packages, forming a package hierarchy. The visibility of a term is controlled in order to realize partial knowledge hiding in the process of knowledge sharing. The whole ontology is composed of a set of packages. A concrete package-based ontology language, i.e., the package-based partial-order (hierarchy) ontology (PPO), is studied to illustrate the features of such modular ontologies. We also present a modular ontology editor for building PPO and we show its applications on practical ontologies.

1 Introduction

Semantic Web [3] aims to support seamless and flexible access and use of semantically heterogeneous, networked data, knowledge, and services. The success of the Semantic Web relies on the availability of a large collection of domain or application specific ontologies and mappings between ontologies to allow integration of data [16]. An increasing need for sharing information and services between autonomous organizations has led to major efforts aimed at the construction of ontologies in many domains e.g., the gene ontology (GO, www.geneontology.org).

With the rapid proliferation of ontologies, their complexity (e.g., size, structure) and the complexity of the process of building ontologies are also increasing. A typical ontology used today contains thousands of terms. For example, GO contains 2×10^5 terms and the Gramineae Taxonomy Ontology contains 7×10^5 terms.

Large-scale ontologies bring new challenges for the building, adaption, use and reuse of such ontologies. There is a great need for special mechanisms and tools for well-controlled and efficient management of large ontologies.

By its very nature, ontology construction is a *collaborative* process which involves direct cooperation among individuals or groups of domain experts or indirect cooperation through the reuse or adaptation of previously published, autonomously developed, very likely, semantically heterogeneous ontologies. In order for an ontology to be broadly useful to a certain community, it needs to capture the knowledge based on the collective expertise of multiple experts and research groups. Typically, a large ontology is built and curated by a community. Each member of the community only contributes a part of the ontology and the final ontology is compiled from the contributed pieces.

The mechanisms and tools should be *scalable* to handle large-scale ontology for storage, editing, browsing, visualizing, reasoning and reusing. While a small or moderate-sized ontology can be fully loaded into an ontology tool (e.g., editor or reasoner), it's usually very inefficient or even impossible to fully load a large ontology into such a tool. We need tools that are able to process large ontologies with limited time and space (e.g., memory) resources.

Hence, there is an acute need for approaches and tools that facilitate collaborative and scalable design and reuse of ontologies. Against this background, this paper describes a modular ontology approach to the problem described and studies the package-based organization and development of ontologies. In particular, we study the package-based partial-order ontology language and tools for modular hierarchies, and illustrate their usefulness with an example from life science.

2 Desiderata of Modular Ontology

Typically, during the construction of a large ontology, multiple relatively autonomous groups will contribute parts of such ontology that pertain to their domains of expertise or responsibility. For example, an ontology about a university contains information about buildings, people, departments, institutions, teaching activities, research activities, finance, administration and so on. The ontology is built by multiple groups of people, each from a relatively autonomous unit, such as different departments, laboratories or offices. The ontology in question should be a semantically coherent integration of the constituent parts developed by the individual groups. We enumerate below, some desiderata of such collaborative large-scale ontology building process.

Local Terminology: Terms used in different ontologies should be given unique identifiers. This is necessary to avoid name conflicts when merging ontologies and to avoid unwanted interactions among modules. For example, *TurkeyStudy* means the study of a bird in the animal science department ontology, while *TurkeyStudy* means the study of a country in the geography science department ontology. Manual processing of such name conflicts does not scale up with increase in size, number, and complexity of ontologies.

Localized Semantics: Ontology construction requires different groups to adapt or reuse ontologies developed by other groups. However, unrestricted use of entities and relationships from different ontologies can result in serious semantic conflicts, especially when the ontologies in question represent local views of the ontology producers. For example, the computer science department may have no qualifying exam for PhD students, therefore, the CS ontology assumes that all *PhDStudent*s are *PhDCandidate*s, while in the electrical engineering department, which requires qualifying exam for all PhD students, *PhDStudent*s includes both *PhDCandidate*s and *ProPhD*.

Partial Reuse: In collaborative design of ontologies, it often makes sense to reuse parts of existing ontologies. However, lack of modularity and localized semantics in ontologies forces an all or nothing choice. For example, a department library may want to reuse only relevant part of the Congress Library Catalog (CLC) ontology in creating its own ontology. Nevertheless, current ontology languages such as OWL do not support such partial importing of an ontology. Modular ontologies will facilitate more flexible and efficient reuse of existing ontologies. For example, the computer science department library may just reuse the Q76 component of the CLC ontology.

Knowledge Hiding: In many applications, the provider of an ontology may not wish, because of copyright considerations or privacy or security concerns, to make the entire ontology visible to the outside, while willing to expose certain parts of the ontology to certain subsets of users. For example, the SSN information of the university employees should be hidden to general access, while visible to a particular component, such as the payroll office ontology. In other cases, an ontology component may only provide a limited query interface, while the details of the component are not important or necessary to users and other ontology components. For example, the department ontology may share course grading information to the registration office, but hold detailed homework and exam grading records to local access only.

Ontology Evolution: Ontology construction is usually an iterative process. A small change in one part of an ontology may be propagated in an unintended and hence undesirable manner across the entire ontology. For example, a term can be obsoleted (such as a graduated student name or a canceled course name) but still be referred by other parts. A concept definition can be changed without informing other parts, for instance, the computer science department may introduce the PhD qualifying exam this year and change its definition of *PhDStudent* to $PhDCandidate \sqcup ProPhD$, while some other parts still hold to its old definition $PhDStudent \equiv PhDCandidate$.

Distinction between Organizational and Semantic Structure: Two kinds of knowledge structure could be recognized in a knowledge base. One is organizational structure, which arranges terms for better usage and understanding. For example, we can have a complete, huge English dictionary, or instead, we can have domain-specific dictionaries, such as computer science dictionary and life science dictionary. The second kind of structure is the semantic structure, which has to do with how the meaning of terms is related. For example, 'Mouse' is an 'Animal' or 'Mouse' is part of a 'PC'. In the university ontology example, we may compile the university people directory from multiple department directories (the organizational structure), and pertain the classification of the people such as 'Alice' is a 'Student' (the semantic structure).

Proposed Approach

Current ontology languages, like OWL, while they offer some degree of modularization by restricting ontology segments into separated XML namespaces, fail to fully support modularity, localized semantics, and knowledge hiding. This state of affairs in ontology languages is reminiscent of the early programming languages and first attempts at software engineering when uncontrolled use of global variables, spaghetti code, absence of well-defined modules were leading to unwanted and uncontrolled interactions between code fragments.

In this paper, we argue for package based ontology languages to overcome these limitations. A package is an ontology module with clearly defined access interface. Mapping between packages is performed by views, which define a set of visible terms on the referred packages. Semantics are localized by hiding semantic details of a package by defining appropriate interfaces (special views). Packages provide an attractive way to compromise between the need for knowledge sharing and the need for knowledge hiding in collaborative design and use of ontologies. The structured organization of ontology entities in packages bring to ontology design and reuse, the same benefits as those provided by packages in software design and reuse in software engineering.

3 Package-based Ontologies

This section explores the basic definitions of package-based ontology.

3.1 General Ontology

In knowledge engineering, an ontology is the shared specification of a conceptualization [13]. Here we first give a general definition of an ontology.

Definition 1 (Ontology) *An ontology is a tuple* (S, R)*, where* $S = S_I \cup S_C \cup S_P$ *is the set of terms and* S_I, S_C, S_P *are the set of individuals, concepts, and properties, respectively,* $R = S \times S$ *is the set of associations of the terms.*

An interpretation of an ontology is a tuple (Δ, Φ)*, where* Δ *is the domain of all possible individuals, a subset of* Δ *is the interpretation of a concept,* $\Phi \subseteq \Delta^n, n \geqslant 2$ *is the domain of all properties. The interpretation of a term* t *is denoted as* t^I*.*

For example, for an ontology (S, R), where $S = \{Alice, Bob, Student, People, Knows\}$, $R = \{Student \sqsubseteq People,\ Alice\ isa\ Student,\ Bob\ isa\ Student\}$, $S_I = \{Alice, Bob\}$ is the individual set, $S_C = \{Student, People\}$ is the concept set, $S_P = \{Knows\}$ is the property set.

In an interpretation of the ontology, concepts are subsets of the domain, e.g. $Student^I \subset \Delta$, individuals are elements in the domain, e.g. $Alice^I \in \Delta$, $Bob^I \in \Delta$, properties are Cartesian products of the domain, e.g. $Knows^I \subset People^I \times People^I$. Association $Student \sqsubseteq People$ is interpreted as $Student^I \subseteq People^I$, $Alice\ isa\ Student$ as $Alice^I \in Student^I$.

A specific ontology language, such as the description logics $\mathcal{SHIQ(D)}$, can be extended from the general ontology definition. In the next section, we will study partial-order ontology as a concrete example.

3.2 Package and Package Hierarchy

In this research, we introduce **package** [2] as the basic module of an ontology. In such a modular ontology representation, an ontology is divided into smaller components called packages, and each package contains a set of highly related terms and their relations; packages can be nested in other packages, and form a package hierarchy; the visibility of a term is controlled by scope limitation modifiers, such as `public`, `private` and `protected`. The whole ontology is composed of a set of packages.

We first give the definition of package.

Definition 2 (Package) *A package* $P = (S_P, R_P)$ *of an ontology* $O = (S, R)$ *is a fragment of* O*, such that* $S_P \subseteq S$*,* $R_P \subseteq R$*. The set of all possible packages is denoted as* Δ_P*.*

A term $T \in S_P$ *is called a* member *of* P *and can also be denoted as* $T \in P$*.* P *is called the* home package *of* T *and denoted as* $P = \mathcal{HP}(T)$*.*

Global package P_{global} *is a special package which is the default package for any term without package membership.*

For example, an ontology O has two packages $P_{student}$ and $P_{faculty}$. $P_{student}$ has terms $Alice, Bob, Student, People$, $P_{faculty}$ has terms $Carl, Dell, Faculty$. $Alice$ is a member of $P_{student}$, and $P_{student}$ is the home package of $Alice$: $P_{student} = \mathcal{HP}(Alice)$. Both $P_{student}$ and $P_{faculty}$ are elements of the package domain Δ_P.

A package can be declared as a sub package of another package. Therefore, the whole ontology will have an organizational hierarchy, in additional to the semantic hierarchy. Formally, we define package nesting as

Definition 3 (Package Nesting) *A package* P_1 *can be called nested in another package* P_2 *and denoted as* $P_1 \in_N P_2$*. All package nesting relations in an ontology form the* organizational hierarchy *of the ontology.*

Transitive nesting $\in_N^*$ *is defined as*

- $P_1 \in_N P_2 \to P_1 \in_N^* P_2$
- $P_1 \in_N^* P_2$ *and* $P_2 \in_N^* P_3 \to P_1 \in_N^* P_3$

One advantage of package hierarchy is that both concepts and individuals of a module can be structured in an organizational hierarchy, while their semantics could have different hierarchy or no hierarchy at all. For example, the $P_{student}$ package may include two sub packages $P_{cs-student}$ and $P_{ee-student}$, developed by the computer science department and the electrical engineering department respectively. Part of the contents of these packages is shown in Figure 1. We can see that the concept semantic hierarchy is different to the package organizational hierarchy, and individuals have organizational hierarchy but no semantic hierarchy.

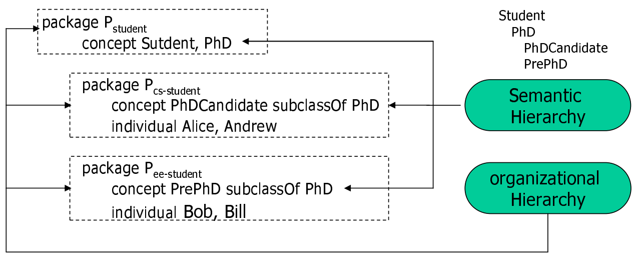


Figure 1. Organizational Hierarchy and Semantic Hierarchy

3.3 Term Scope Limitation

A term defined in a package can also be associated with a scope limitation modifier (SLM). SLM of a term controls the visibility of the term to other packages. For example, a term with SLM 'public' can be visited from any package (See Figure 2). Formally, a SLM is defined as follows.

Definition 4 (SLM) *The scope limitation modifier of a term t in a package P is a boolean function $V(p,t)$, where p is another package, and p can access t iff $V(p,t) =$ **TRUE**. We denote $t \in_V P$.*

We define three default SLMs as

- $public(p,t) :=$ **TRUE**, means term t is accessible everywhere.
- $protected(p,t) := (t \in p)$ or $p \in^*_N \mathcal{HP}(t)$, means t is visible to its home package and all its descendant packages on the organizational hierarchy.
- $private(p,t) := (t \in p)$, means t is visible only to its home package.

We can also define other types of SLMs as needed. For example, $friend(P,t) :=$ **TRUE** will grant access of t to a particular package P.

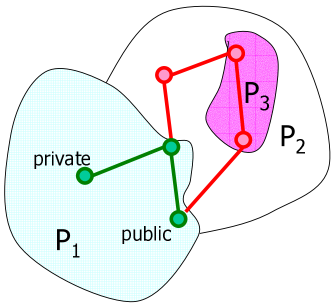


Figure 2. Package-based Ontology: The ontology has three packages P_1, P_2, P_3. P_3 is nested in P_2. A public term in P_1 is visible to P_2 while a private term in P_1 is only visible in the home package(P_1).

When SLMs are also included in the package definition, a package is defined as (S_P, R_P, SLM_P) where for any $t \in S_P$, there is one and only one SLM $\in SLM_P$ for t.

The *scope* of a term t is the set of packages from where t is visible, i.e.

$$scope(t) = \{p | SLM(p,t) = \textbf{TRUE}\}.$$

The horizon of a package p is the set of all terms that are visible to p, i.e.

$$horizon(p) = \{t | SLM(p,t) = \textbf{TRUE}\}$$

Scope limitation can serve for multiple purposes, such as

- Hide private or copyrighted information. For example, the $P_{cs-student}$ package may include SSN information which is not shared except to particular friend packages (such as $P_{registration}$)

- Restrict extensibility and reduce unwanted coupling. When different packages are defining the ontology on different level of abstraction, the high-level package may publicize its 'open' terms (the terms that need further specialization) to low-level packages, and keep 'final' terms (the terms that don't need extension) local, and therefore reduce the possibility of unintended coupling. For example, in the *design* stage of the university ontology, to build the administration hierarchy of university units, the high level package P_{adm} may make 'President' as final term while 'DepartmentHead' as open term, the lower level package P_{cs-adm} may extend 'DepartmentHead' with 'CsDepartmentHead' as final term, and further keep 'CsLabDirector' as open term.

- Hide details to improve usability. For example, $P_{student}$ defines property $CourseGrading$ as public but keeps $HomeworkGrading$ open only to sub packages and $P_{registration}$.

3.4 Views and Interfaces

Views and interfaces provide a flexible way to connect and reuse packages. A view is a set of visible terms from one or more packages. An interface is a view over only one package. Formally we have

Definition 5 (View and Interface) *A* view *V for a set of packages $P_1, ...P_n$, denoted as $P_1, ...P_n :: V$, is a term set $V \subseteq \{t | \forall p, SLM(p, t) = \textbf{\textit{TRUE}}, t \in P_i, 1 \leqslant i \leqslant n\}$. A view is an interface if $n = 1$.*

The default interface *of a package P is the public term set of the package, i.e. $\{t | t \in_{public} P\}$.*

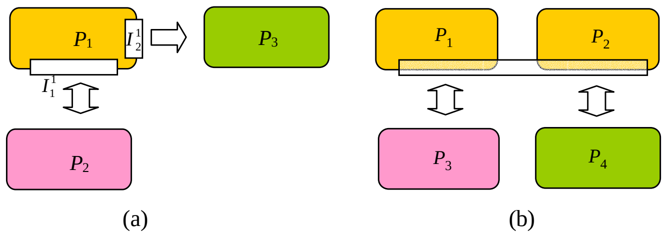


Figure 3. View and Interface: (a) A Package with multiple interfaces (b) A View can be built upon multiple packages and can be referred by multiple modules

We can benefit from views and interfaces in the following ways

- Views can help to *integrate* ontology components (Figure 3 (b)). The terms defined in a view can be from different packages, therefore providing a way to combine ontology pieces. For example, a $V_{cs-people}$ view can be defined to unify information from both $P_{cs-student}$ and $P_{cs-faculty}$.

- Views can be used to 'customize' packages, i.e. have *polymorphism* (Figure 3 (a)). Packages can have multiple views for different purposes. For example, the terms in package $P_{cs-student}$ can be included in the view $V_{cs-people}$ (the people in the computer science department) or $P_{student}$ (students in the university) for different usages.

- Packages can be *partially reused* with interfaces. An interface contains a selected set of terms defined in a package, therefore enabling the reuse of the package with reduced complexity and focused topics. For example, the publication ontology module of the computer science department (as a part of the research activities ontology module) contains the bibtex information for all publications over the years in the department. As the module (as package P_{cs-pub}) may contain thousands of terms, it may also provide specialized interfaces for different topics, such as software engineering (with interface $V_{cs-pub-se}$) or artificial intelligence (with interface $V_{cs-pub-ai}$).

- Interfaces can help in *ontology evolution*. In the design of an ontology, the details of an ontology module may change over the time. However, with a well-defined interface, such changes can be hidden to other modules or users with the separation between the change-prone terms and the stable terms. For example, the $P_{cs-student}$ package may contain a interface $V_{cs-current-student}$, which contains current students in the department. While the current student list is ever-changing and the definition of 'current student' might change (e.g., excludes minor degree students), the interface can be kept the same.

The aforementioned definition of package-based ontology is summarized in the Table 1. Δ_S is the ontology term domain, which is the set of all possible names in the ontology. Δ_P is the domain of all possible packages.

4 Package-based Partial-order Ontology (PPO)

In this section, we will study a concrete package-based ontology language to show how to extend an existing ontology language to a modular ontology language. We choose the partial-order ontology (PO) as the example. PO is relatively simple but representative and widely applicable, since it can be used as the theoretical model for the most popular ontologies, i.e., hierarchies.

4.1 Partial-order ontology

Firstly we give the standard definition of partial-order and partial-order ontology.

Definition 6 (Partial-Order) *A partial-order $\leq$ over a set S is a relation over $S \times S$ such that*

- *$\leq$ is transitive : $x \leq y$ and $y \leq z \rightarrow x \leq z$*

- *$\leq$ is self-reflexive: $x \leq x$*

Table 1. Syntax and semantics of Package-based Ontology

Constructor	Syntax	Semantics
Package	P	$P^I \in \Delta_P$
Global Package	P_{global}	$P^I_{global} \in \Delta_P$
Membership	$t \in P$ or $member(t, P)$	$member^I \subseteq \Delta_S \times \Delta_P$
Home Package	$\mathcal{HP}(t)$	$\mathcal{HP}(t)^I = p$, where $t \in p$
Nesting	$\in_N$	$\in^I_N \in \Delta_P \times \Delta_P$
Transitive nesting	$\in^*_N$	$\in_N \rightarrow \in^*_N$, $\in^*_N = (\in^*_N)^+$
SLM	SLM(p,t)	$p \in \Delta_P$ can access $t \in \Delta_S$ iff SLM(p,t)=**TRUE**
	public(p,t)	$\forall p, public(p,t) :=$ **TRUE**
	private(p,t)	$\forall p, private(p,t) := (p = \mathcal{HP}(t))$
	protected(p,t)	$\forall p, protected(p,t) := (p = \mathcal{HP}(t)$ or $p \in^*_N \mathcal{HP}(t))$
View	$P_1, ...P_n :: V$	$V^I \subseteq \{t \mid \forall p, SLM(p,t) = \textbf{TRUE}, t \in P_i, 1 \leqslant i \leqslant n\}$
Interface	$P :: F$	$F^I \subseteq \{t \mid \forall p, SLM(p,t) = \textbf{TRUE}, t \in P\}$

- *$\leq$ is anti-symmetric: $x \leq y$ and $y \leq x \rightarrow x = y$*

Definition 7 (Partial-Order Ontology) *A partial-order ontology over a set S has*

- *A set of* ***partial-order axioms*** *$x \leq y$, where $x, y \in S$;*
- *A set of* ***equivalence axioms*** *$x = y$ or* ***nonequivalence axioms*** *$x \neq y$.*

Each item in S is called a term.

A partial-order ontology can be mapped to a graph with each term as a node and each partial-order axiom as a directed edge. Equivalent terms share the same node. In particular, a PO ontology can be a hierarchy, i.e., a Directed Acyclic Graph (DAG), which has no loops. A DAG is a tree if all nodes except one (the root, which has no incoming edge) has one and only one incoming edge.

For example, we can describe professions with a hierarchy such as:

```
Faculty
   Professor
   ...
Student
   PhDStudent
   MasterStudent
   Undergraduate
Staff
   Secretary
```

where indent means a partial-order, e.g. Professor $\leq$ Faculty.

4.2 Package-based Partial-order ontology

A modular partial-order ontology has one or many packages. Each package in the ontology is a partial-order ontology component.

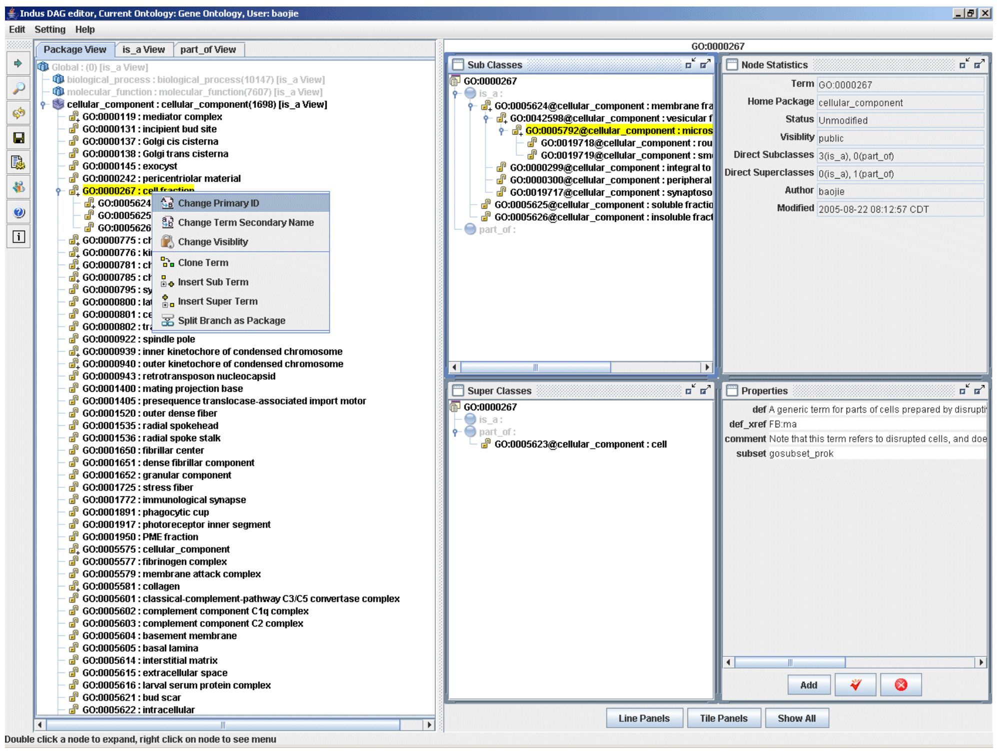
Definition 8 (Package-based PO ontology) *A package-based partial-order ontology is a set of related packages $\{P_1, ...P_n\}$, where $P_i = (S_i, R_i)$, S_i is a term set and R_i is a set of partial-order, equivalence or nonequivalence axioms.*

Packages in a PPO can have different roles (see Figure 4), such as

- **Branch Packages** A big hierarchy may be divided into smaller branches and each of them is a sub-package. For example, to build the administration hierarchy, we may create high level topic packages, such as P_{adm}, and then define nested packages such as P_{cs-adm} or P_{ee-adm}. Here, different level packages capture different levels of abstraction of concepts.
- **Aspect Packages** An ontology may contain hierarchies for different aspects of the domain. For example, in additional to the administration hierarchy, we may also describe people with the profession hierarchy.

4.3 Building PPO

We developed the INDUS DAG Editor, a modular ontology editor, to enable the building and deployment of PPO. The editor enables the ontology developer to create a community-shared ontology server with



```
Global : (0) [is_a View]
  biological_process : biological_process(10147) [is_a View]
  molecular_function : molecular_function(7607) [is_a View]
  cellular_component : cellular_component(1698) [is_a View]
    GO:0000119 : mediator complex
    GO:0000131 : incipient bud site
    GO:0000137 : Golgi cis cisterna
    GO:0000138 : Golgi trans cisterna
    GO:0000145 : exocyst
    GO:0000242 : pericentriolar material
    GO:0000267 : cell fraction
      GO:0005624
      GO:0005625
      GO:0005626
    GO:0000775 : ch
    GO:0000776 : ki
    GO:0000781 : ch
    GO:0000785 : ch
    GO:0000795 : sy
    GO:0000800 : la
    GO:0000801 : ce
    GO:0000802 : tr
    GO:0000922 : spindle pole
    GO:0000939 : inner kinetochore of condensed chromosome
    GO:0000940 : outer kinetochore of condensed chromosome
    GO:0000943 : retrotransposon nucleocapsid
    GO:0001400 : mating projection base
    GO:0001405 : presequence translocase-associated import motor
    GO:0001520 : outer dense fiber
    GO:0001535 : radial spokehead
    GO:0001536 : radial spoke stalk
    GO:0001650 : fibrillar center
    GO:0001651 : dense fibrillar component
    GO:0001652 : granular component
    GO:0001725 : stress fiber
    GO:0001772 : immunological synapse
    GO:0001891 : phagocytic cup
    GO:0001917 : photoreceptor inner segment
    GO:0001950 : PME fraction
    GO:0005575 : cellular_component
    GO:0005577 : fibrinogen complex
    GO:0005579 : membrane attack complex
    GO:0005581 : collagen
    GO:0005601 : classical-complement-pathway C3/C5 convertase complex
    GO:0005602 : complement component C1q complex
    GO:0005603 : complement component C2 complex
    GO:0005604 : basement membrane
    GO:0005605 : basal lamina
    GO:0005614 : interstitial matrix
    GO:0005615 : extracellular space
    GO:0005616 : larval serum protein complex
    GO:0005621 : bud scar
    GO:0005622 : intracellular
```

GO:0000267

Sub Classes

```
GO:0000267
  is_a :
    GO:0005624@cellular_component : membrane fra
      GO:0042598@cellular_component : vesicular f
        GO:0005792@cellular_component : micros
          GO:0019718@cellular_component : rou
          GO:0019719@cellular_component : sm
      GO:0000299@cellular_component : integral to
      GO:0000300@cellular_component : peripheral
      GO:0019717@cellular_component : synaptoso
    GO:0005625@cellular_component : soluble fractio
    GO:0005626@cellular_component : insoluble fract
  part_of :
```

Node Statistics

Term	GO:0000267
Home Package	cellular_component
Status	Unmodified
Visiblity	public
Direct Subclasses	3(is_a), 0(part_of)
Direct Superclasses	0(is_a), 1(part_of)
Author	baojie
Modified	2005-08-22 08:12:57 CDT

Super Classes

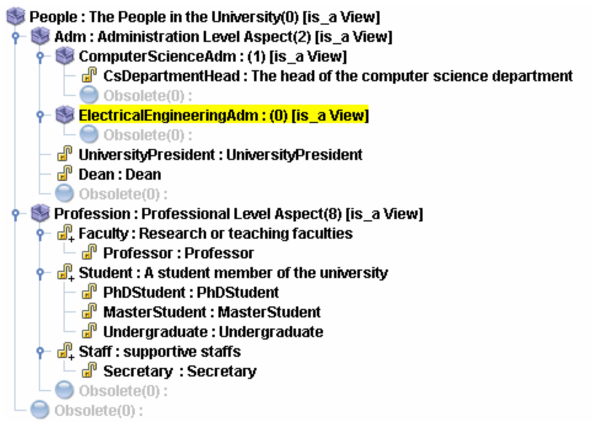
```
GO:0000267
  is_a :
  part_of :
    GO:0005623@cellular_component : cell
```

Properties

def	A generic term for parts of cells prepared by disrupti
def_xref	FB:ma
comment	Note that this term refers to disrupted cells, and doe
subset	gosubset_prok

Double click a node to expand, right click on node to see menu

Figure 5. the INDUS DAG Editor, a collaborative editor for package-based partial-order ontology



```
People : The People in the University(0) [is_a View]
  Adm : Administration Level Aspect(2) [is_a View]
    ComputerScienceAdm : (1) [is_a View]
      CsDepartmentHead : The head of the computer science department
      Obsolete(0) :
    ElectricalEngineeringAdm : (0) [is_a View]
      Obsolete(0) :
    UniversityPresident : UniversityPresident
    Dean : Dean
    Obsolete(0) :
  Profession : Professional Level Aspect(8) [is_a View]
    Faculty : Research or teaching faculties
      Professor : Professor
    Student : A student member of the university
      PhDStudent : PhDStudent
      MasterStudent : MasterStudent
      Undergraduate : Undergraduate
    Staff : supportive staffs
      Secretary : Secretary
    Obsolete(0) :
  Obsolete(0) :
```

Figure 4. Example of Package-based Partial-Order Ontology

database storage, which supports concurrent browsing and editing of the ontology. The editor supports team collaboration with features such as:

- The ontology is stored on a relational database server; 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 editing.
- The editor supports multi-relation hierarchy. For example, the ontology can have both is-a and part-of hierarchies while some terms are shared in both hierarchies. Such structure is widely used in life science ontologies.
- The editor supports multiple users to concurrently edit an ontology. A locking mechanism is provided to avoid conflicts and abnormalities. Modules of the ontology can be developed by different authors and assembled at latter stages.
- User profile management is provided. Authors of the ontology have different levels of privileges (such as ontology admin and package admin) over modules in that ontology. The author of a package can authorize other users the access to certain terms, therefore controls the extendibility of that package.
- The editor provides a handy graphical user interface (GUI) to edit and browse a DAG.
- The editor can import and export ontologies in standard format, such as OBO (Open Biomedical Ontologies) and OWL (Web Ontology Language).

In Figure 5, we show a snapshot of the editor loaded with the Gene Ontology (GO). Three namespaces of

GO is modeled as three top packages. Each package can be viewed in either is-a view or part-of view. The neighbor terms of a selected term on both hierarchies are shown in the smaller browsing panels. A user can open one package instead of the whole ontology to reduce the memory requirement.

4.4 Design Principles of PPO

In the practice of package-based ontology building, we get some experience on the design of modular hierarchies.

Moderate Size. A package should not be too small or too big. Typically a package is developed by an autonomous author and the author should have good knowledge about the package. A large package is beyond the control of a single author, therefore it is not safe on consistency and inefficient for loading and reasoning. We are working on a package decomposition method to limit the size of a single package. A small package is also inefficient as a piecemeal module.

Overlap the package hierarchy and the Semantic hierarchy if you can. It is best to keep the package hierarchy consistent with the semantic hierarchy (such as isa hierarchy), since it will be more semantically clear and safe. High level packages should contain general concepts, while low level packages should contain specialized concepts.

Reuse 'Pattern'. If a 'pattern' is repeated in different packages, it is better to extract it as a reusable package. For example, if the hierarchy *DepartmentHead - LabDirector -ResearchAssistant* repeats in many departments, it can be extracted in a common template package for all departments.

Term scope limitation can be changed in the released ontology. The use of term SLM in the design stage and the release stage can be different. In the design stage, we are focused on avoiding name conflicts, reducing unwanted coupling, less memory demand and good evolvability, while in the release stage, we are focused on partial re-usability and critical information hiding. For example, we have mentioned in 3.3 that a P_{adm} may make '*President*' as a private final term to restrict its extensibility. We can change it to a public term in the released ontology.

Package-based ontology can be reduced to normal *ontology.* If partial reusing and information hiding are not concerned issues, the package-based ontology can be reduced to a 'normal' ontology without packages in the released version. For example, even if we edit GO with package-based structure, to release it in generally acceptable format, the released ontology can contain only the semantic structure of the ontology, but no package organization or scope limitation. This is a way to comprise the modular ontology building and current non-modular ontology usage.

5 Relation Work

Distributed Logics: A number of distributed logics systems have been studied during recent years. Examples include Local Model Semantics [10] and Distributed First Order Logic (DFOL) [11] which emphasize local semantics and the compatibility relations among local models. Inspired by DFOL, Borgida *et.al.* [4] extends the description logic to obtain a distributed description logic (DDL) system. A DDL system consists of a set of distributed TBoxes and ABoxes connected by "bridge rules". Bridge rules are unidirectional, thereby ensuring that there is no "back-flow" of information among modules being connected by a bridge rule. The authors in [15] extend local model semantics and harmonize local models via agreement on vocabulary provenance.

Grau *et.al.* [12] explores using $\mathcal{E}$-connections to extend OWL and this approach is very straightforward to implement on existing tableau OWL reasoners.

Serafini *et.al.* [17, 18] defines a sound and complete distributed tableau-based reasoning procedure which is built as an extension to standard Description Logic tableau. [18] also describes the design and implementation principles of a distributed reasoning system, called DRAGO (Distributed Reasoning Architecture for a Galaxy of Ontologies), that implements such distributed decision procedure.

Modular Ontologies Two approaches to integration of separate ontologies have been developed based on DFOL and DDL. The Modular Ontology [19] offers a way to exploit modularity in reasoning. It also defines an architecture that supports local reasoning by compiling implied subsumption relations and offers a way to maintain the semantic integrity of an ontology when it undergoes local changes. In the "view-based" approach to integrating ontologies, all external concept definitions are expressed in the form of queries. However, A-Box is missing in the query definition, and the mapping between modules is unidirectional, making it difficult to preserve local semantics.

Fikes *et.al.* [9] mentioned integration of modular ontology in the Ontolingua system, which restricts symbol access to public or private. The major difference between our approach and their approach is that we use packages not only as modular ontology units, but also in organizational hierarchies, therefore enabling the hierarchical management of modules in collaborative ontology building. The scope limitation modifier idea is an extension of the symbol access restriction, but it is more flexible and expressive.

Contextual Ontology: Contextual logic, a formalism based on DDL, emphasizes localized semantics in ontologies. Contextual ontology keeps contents local and maps the content to other ontologies via explicit bridge rules. Bouquet *et.al.* [5] proposed CTXML, which includes a hierarchy-based ontology description and a context mapping syntax. They further [6] combined CTXML and OWL into Context OWL (C-OWL), a syntax for bridge rules over OWL ontology. Our approach has several improvements over C-OWL by introducing scope limitation modifiers (SLM). Bridge rules can be viewed as special cases of queries and SLM offers a controllable way to keep content local by definition.

Serafini *et.al.* [14] gave a survey of existing ontology mapping languages, such as DDL, C-OWL, OIS [7], DLII [8], and $\mathcal{E}$-connections, by translating them into distributed first order logic. None of those approaches provides scope limitation, therefore it is limited on ontology partial reuse and avoidance of unintended coupling.

6 Conclusion

Modularity in ontologies is beneficial to ontology engineering in several ways. It simplifies the maintenance and evolution of ontologies and mappings between ontologies; it enables flexible partial reuse of a large ontology; it offers a more efficient organization for an ontology; it also improves the processing time when querying ontology.

In this research, we introduce **package** as the basic module of ontologies. In such a modular ontology representation, an ontology is divided into smaller components called packages, and each package contains a set of highly related terms and their relations; packages can be nested in other packages, forming a package hierarchy; the visibility of a term is controlled by scope limitation modifier such as `public`, `private` and `protected`. The whole ontology is composed of a set of packages. We have shown as an example a package-based partial-order ontology and described the implementation of a collaborative ontology building tool for such ontologies.

The major contributions of the paper are

- The formal definition of a package-based ontology as a modular ontology approach for large-scale ontology building and using.
- The proposal of scope limitation of ontology terms to support partial knowledge hiding in knowledge sharing. Such design is beneficial to avoid unintended coupling and to keep private information.
- The implementation of a modular ontology editor for collaborative ontology building with good scalability.

We will continue the work on several directions

- **Extend the language expressiveness**. We have proposed a preliminary framework to extend description logics (DL) with packages [1]. For example, the well-known DL wine ontology[1] can be divided into several packages, such as a *Region* package, a *Food* package and a *Wine* package (inside *Food*). It will help to partially reuse the ontology, for example, to reuse the *Region* ontology in other applications. In the definition of grape, fine-grain concept like *CabernetFranceGrape*, which is only interesting in winery but not to other domains, can be declared as 'protected' in *Wine*. We will study the syntax and semantics of package-based description logics as well as the reasoning task for such ontologies.
- **The distributed reasoning algorithm**. Reasoning with package-based ontologies is a special case of distributed ontology reasoning. The native reasoning ability is provided by local modules and the total reasoning process is based on these native reasoners. We will study reasoning algorithms for PPO. Since the basic reasoning task in a single DAG can be reduced to a graph reachability problem or the transitive closure computation problem, the reasoning task of PPO will be a distributed version of such problems. Also, because modular ontologies have special properties compared to general graphs (for example, limited depth and limited branch factor), we may find optimized algorithms with better performance on time complexity and/or space complexity.
- **Improve the modular ontology editor**. We will extend the existing tool to more expressive languages such as OWL and connect the modular ontology reasoner with the editor to ensure consistency of the ontology.

Acknowledgment

This research is supported in part by grants from the National Science Foundation (0219699) and the National Institutes of Health (GM 066387) to Vasant Honavar

[1]http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine#

References

[1] J. Bao and V. Honavar. Ontology language extensions to support localized semantics, modular reasoning, and collaborative ontology design and ontology reuse. Technical report, TR-341, Computer Sicence, Iowa State University, 2004.

[2] J. Bao and V. Honavar. Ontology language extensions to support localized semantics, modular reasoning, collaborative ontology design and reuse. In *3rd International Semantic Web Conference (ISWC2004), Poster Track , 7-11 November 2004, Hiroshima, Japan*, 2004.

[3] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.

[4] A. Borgida and L. Serafini. Distributed description logics: Directed domain correspondences in federated information sources, 2002.

[5] P. Bouquet, A. Dona, L. Serafini, and S. Zanobini. Conceptualized local ontologies specification via ctxml. In *Working Notes of the AAAI-02 workshop on Meaning Negotiation, Edmonton (Canada).*

[6] P. Bouquet, F. Giunchiglia, and e. F. van Harmelen. C-OWL: Contextualizing ontologies. In *Second International Semantic Web Conference*, volume 2870 of *Lecture Notes in Computer Science*, pages 164–179. Springer Verlag, 2003.

[7] D. Calvanese, G. D. Giacomo, and M. Lenzerini. A framework for ontology integration. In *The Emerging Semantic Web*, 2001.

[8] D. Calvanese, G. D. Giacomo, and M. Lenzerini. Description logics for information integration. In *Computational Logic: Logic Programming and Beyond*, pages 41–60, 2002.

[9] R. Fikes, A. Farquhar, and J. Rice. Tools for assembling modular ontologies in ontolingua. In *AAAI/IAAI*, pages 436–441, 1997.

[10] C. Ghidini and F. Giunchiglia. Local model semantics, or contextual reasoning = locality + compatibility. *Artificial Intelligence*, 127(2):221–259, 2001.

[11] C. Ghidini and L. Serafini. *Frontiers Of Combining Systems 2, Studies in Logic and Computation*, chapter Distributed First Order Logics, pages 121–140. Research Studies Press, 1998.

[12] B. C. Grau, B. Parsia, and E. Sirin. Working with multiple ontologies on the semantic web. In *International Semantic Web Conference*, pages 620–634, 2004.

[13] T. R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, 1993.

[14] H. S. Luciano Serafini and H. Wache. A formal investigation of mapping language for terminological knowledge. In *19th IJCAI*, 2005.

[15] Y. Qu and Z. Gao. Interpreting distributed ontologies. In *Alternate track papers & posters of the 13th international conference on World Wide Web*, pages 270–271. ACM Press, 2004.

[16] J. Reinoso-Castillo, A. Silvescu, D. Caragea, J. Pathak, and V. Honavar. Information extraction and integration from heterogeneous, distributed, autonomous information sources: A federated, query-centric approach. In *IEEE International Conference on Information Integration and Reuse*, 2003.

[17] L. Serafini and A. Tamilin. Local tableaux for reasoning in distributed description logics. In *Description Logics*, 2004.

[18] L. Serafini and A. Tamilin. Drago: Distributed reasoning architecture for the semantic web. In *ESWC*, pages 361–376, 2005.

[19] H. Stuckenschmidt and M. Klein. Modularization of ontologies - wonderweb: Ontology infrastructure for the semantic web.