Towards Collaborative Environments for Ontology Construction and Sharing

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

Abstract— **Ontologies that explicitly identify objects, properties, and relationships of interest in specific domains of inquiry are essential for collaborations that involve sharing of data, knowledge, or resources (e.g., web services) among autonomous individuals or groups in open environments. In such a setting, there is a need for software that supports collaboration among groups with different expertise in developing complex ontologies, selectively sharing or selectively adopting parts of existing ontologies, and for constructing application or domain specific ontologies from a collection of available ontology modules. Against this background, this paper motivates the need for collaborative environments for ontology construction, sharing, and usage; identifies the desiderata of such environments; proposes package-extended description logics (P-DL) that extend classic description logic (DL) based ontology languages to support modularity and (selective) knowledge hiding. In P-DL, each ontology consists of packages (or modules) with well-defined interfaces. Each package encapsulates a closely related set of terms and relations between terms. Together, these terms and relations represent the ontological commitments about a small, coherent part of the universe of discourse (e.g., wine). Packages can be nested in another package, forming a package hierarchy, which offers a way to bring better organizational structure of the ontology. Package-extended ontologies also allow creators of packages to exert control over the visibility of each term or relation within the package, thereby allowing selective sharing (or conversely, hiding) of ontological commitments captured by a package.**

I. Introduction

Ontologies that explicitly identify objects, properties, and relationships of interest in specific domains of inquiry are essential for collaborations that involve sharing of data, knowledge, or resources (e.g., web services) among autonomous individuals or groups in open environments, such as the Semantic Web [5]. Consequently, there has been a significant body of recent work on languages for specifying ontologies, software environments for editing ontologies, algorithms for reasoning with, aligning, and merging ontologies [18]. However, the lack of *collaborative* environments for construction, sharing, and usage of ontologies is a major hurdle to the large-scale adoption and use of ontology-based approaches to sharing of information and resources, which is needed to realize the full potential of Semantic Web.

Semantic Web ontologies or, in general, knowledge bases, have several important characteristics, as follows:

1) Constructing large ontologies typically requires collaboration among multiple individuals or groups with expertise in specific areas, with each participant contributing only a part of the ontology. Therefore, instead of a single, centralized ontology, in most domains, there are multiple distributed ontologies covering parts of the domain.
2) Because no single ontology can meet the needs of all users under every conceivable scenario, the ontology that meets the needs of a user or a group of users needs to be assembled from several independently developed ontology modules. Since different ontologies or different modules of a single ontology are developed by people with diverse points of view, semantic inconsistencies or conflicts between such modules are inevitable. Consequently, in collaborative ontology environments, there is a need for mechanisms for resolving or managing such semantic conflicts to ensure that the resulting ontology is not internally inconsistent.
3) While ontologies are often used to facilitate sharing of knowledge, data, and resources, many real-world scenarios also call for selectively *hiding* certain parts of an ontology (or conversely, selectively sharing certain parts of an ontology). The need for knowledge hiding may arise due to privacy and security concerns, or for managing and knowledge engineering purposes.

In contrast, the current state of the art in ontology engineering is reminiscent of the state of programming languages nearly four decades ago: unstructured, with no support for restricting the scope of variables, and limited or no support for program modules, leading to horrendously complex, hard to maintain, seldom reusable code. This needs to be changed in order for the full potential of the Semantic Web to be realized in practice. We need to come to terms with the characteristics of web ontologies. Specifically, next generation ontology languages need to support collaborative construction, selective sharing and use of ontologies. Against this background, this paper introduces the framework of package-extended ontologies to meet this need.

The rest of the paper is organized as follows: Section II elaborates further on the problems that need to be addressed by collaborative environments for ontology construction, sharing, and use. Section III presents basic elements of package-extended ontologies and, in particular, package-extended description logics (P-DL). Section IV discusses the semantics of P-DL. Section V uses a case study to demonstrate how we can represent ontologies with P-DL. Section VI discusses related work. Section VII summarizes the paper and presents several

ideas for future work.

II. Problem Description

As already noted, ontologies that are useful in practice often need to be assembled by selectively combining parts of interrelated, possibly inconsistent modules. For example, an ongoing effort aimed at developing an animal ontology involves a group of individuals, each focused on a specific sub domain of animal knowledge, such as the general animal knowledge, knowledge about pet animals (dogs, cats, etc.), knowledge about poultry, knowledge about livestock, etc. We use this application to introduce some issues that need to be addressed in such a setting, more precisely than was done in the preceding introduction.

A. Local Semantics vs. Global Semantics

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 general animal ontology module may assert that *a dog is a carnivore*, *a carnivore only eats animals*, and *an animal is not a plant* (given in description logic):

$$Dog \sqsubseteq Carnivore$$
$$Carnivore \sqsubseteq \forall eats.Animal$$
$$Animal \sqsubseteq \neg Plant$$

However, in the pet ontology module it asserts that *a sick dog sometimes eats grass*, which *is plant*:

$$SickDog \sqsubseteq Dog \sqcap \exists eats.Grass$$
$$Grass \sqsubseteq Plant$$

There is an inconsistency if the two modules are integrated without proper reconciliation of the semantic conflicts. Each module represents what is believed to be true *from a local point of view* and is *locally consistent*. However, their combination is *not globally consistent*. It is unrealistic to expect that the author of the general animal ontology module can anticipate all possible 'exceptions' that might arise in specific contexts. A potential user of the pet ontology module should not have to discard the general animal ontology module entirely just because of a few inconsistencies that could be managed, if sufficient care is taken to do so.

B. Partial Reuse vs. Total Reuse

In creating a 'MyPet' ontology, one may want to import the knowledge about pets from the comprehensive 'Animal' ontology. However, current ontology languages only allow one to import the 'Animal' ontology in its entirety, although only a small part of it is needed. If an ontology had a modular structure, it would be more flexible and efficient to partially reuse that ontology. Thus, if the 'Animal' ontology was modular, as shown in Fig. 1, only the relevant parts of the whole 'Animal' ontology would be imported into 'MyPet' ontology, thereby avoiding the need to import chunks of unwanted ontology components. This is especially useful when some modules would make 'MyPet' ontology inconsistent if the entire 'Animal' ontology was imported, whereas limiting the reuse of 'Animal' ontology to selected modules would avoid this difficulty.

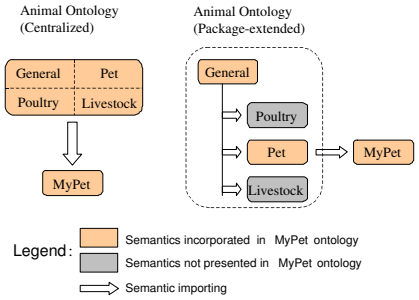

Fig. 1. Total Reuse vs. Partial Reuse

C. Organizational Structure vs. Semantic Structure

It is useful to distinguish between two types of structures in ontologies: organizational structure and semantic structure. The organizational structure consists of an arrangement of terms for better usage and understanding. Domain-specific dictionaries, such as a computer science dictionary or a life science dictionary, where knowledge is *organized* in different modules offer examples of settings where the organizational structure of an ontology is exploited. The semantic structure of an ontology, on the other hand, deals with how the meanings of terms are related: for instance, *mouse is an animal* or *mouse is a part of a computer*. Fig. 2 illustrates the differences between the two types of structures in the case of an animal ontology.

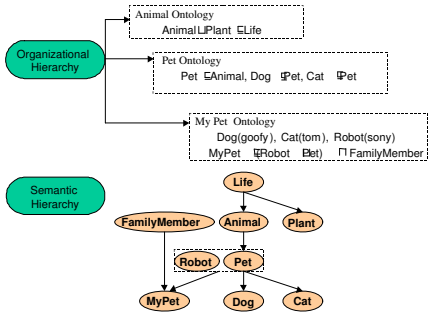

Fig. 2. Organizational Structure vs. Semantic Structure

The distinction between organizational and semantic hierarchies can be understood via analogy with object-oriented programming languages, such as Java and C#. In such languages, new classes can be derived from (and hence semantically related to) existing classes. Such class hierarchies offer an example of semantic structure. Modern object oriented languages like Java also have a notion of packages, which are organized in a package hierarchy. Semantically unrelated classes can be organized into packages that bundle together the classes that are used in a specific class of applications (e.g., graphics).

D. Knowledge Hiding vs. Knowledge Sharing

In many applications, the provider of an ontology may not wish (because of copyright, privacy, security, or commercial concerns) to make the entire ontology visible to the outside,

while being willing to expose certain parts of the ontology to selected subsets of users. In other cases, an ontology component may only provide a limited query interface, as the details of the component are not important to users or to other ontology components. Both cases call for partial knowledge hiding.

For example, the 'Animal' ontology, when completed, may contain a detailed taxonomy of animals. However, the owner of the ontology only exposes a coarse-grained and less professional version to the public, while the fine-grained knowledge is open only for selected scope (e.g., paid users). For instance, the ontology includes such visible and invisible terms and axioms:

visible:
terms: $Dog, Carnivore, Mammals, Animal$
axioms:
$Mammals \sqsubseteq Animal$
invisible:
terms: $Mammalia, Eutheria, Carnivora, Canis, CanisFamiliaris$
axioms:
$Mammalia \equiv Mammals$
$Eutheria \sqsubseteq Mammalia$
$Carnivora \sqsubseteq Eutheria$
$Carnivora \equiv Carnivore$
$Canis \sqsubseteq Carnivora$
$CanisFamiliaris \sqsubseteq Canis$
$Dog \equiv CanisFamiliaris$

The public user may learn by querying the ontology that $Dog \sqsubseteq Carnivore$, $Carnivore \sqsubseteq Animal$ if the inference procedures associated with the ontology can use the invisible (i.e., hidden) part of the ontology in answering user queries, while not exposing the explicit semantics of the hidden axioms. This is an instance of partial knowledge hiding.

Modularity and knowledge hiding are also needed in the case of *semantic encapsulation*. The previous example shows a type of semantic encapsulation where detailed information is hidden in order to provide a simpler query interface. As another example of semantic encapsulation, we can consider a scenario in which two persons, e.g., Alice and Bob, create ontologies for their pets. These ontologies might be queried by software agents of other individuals, e.g., their pet doctors. For example, a doctor may pose a query against the two ontologies, asking if a *pet y has eaten grass*. This query can be denoted by $?(\exists eats.Grass)(y)$ in description logics. There is no requirement that both Alice and Bob use the same type of agents to manage their pet ontologies - after all, they might buy their ontology builders from different software shops - as long as both agents implement the same query interface. For instance, Alice's agent can use a TBox of the ontology language $\mathcal{ALCOI}$

$X \sqsubseteq Grass$, X is nominal
$\{y\} \sqsubseteq Cat$
$\{y\} \sqsubseteq \exists eats.X$

while Bob's agent can use an ontology written in $\mathcal{ALCI}$ with an ABox:

$Grass(x)$
$Dog(y)$
$eats(y, x)$

Both approaches guarantee that the instance class membership query $?(\exists eats.Grass)(y)$ has the same behavior, although the underlying implementations are different. Thus, as the details of the implementations are of no interest to those who query the ontologies, they can be easily hidden from them.

E. Proposed Approach

Current ontology languages like OWL [22] and its corresponding description logics [20], fail to fully support modularity, localized semantics, and knowledge hiding. OWL partially allows ontology modularization and reuse with `owl:imports`. However, the OWL import mechanism has serious drawbacks. First, it directly introduces both terms and semantics of the imported module into the referring modules, therefore providing no way for local semantics. Second, it reveals either all or nothing of a module to another module, therefore supporting neither partial reuse, nor selective knowledge hiding.

Package based ontology language extensions [3], offer a way to overcome these limitations. The resulting ontology language allows the representation of ontology modules using components called packages. Each package typically consists of a set of highly related terms and relationships between them; packages can be nested in other packages, forming a package hierarchy; the visibility of a term is controlled by scope limitation modifiers, such as `public` and `private`. A package has a clearly defined access interface. Semantics can be localized by hiding semantic details of a package by defining appropriate interfaces.

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. Although each package constitutes an internally consistent ontology, there is no requirement that an arbitrary set of packages to be globally consistent. 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.

We introduce package-extended ontologies more precisely in the next section.

III. Package-extended Ontologies

A. Packages as Ontology Organization Units

In a package-extended ontology, the whole ontology is composed of a set of packages. Terms (such as $Dog, Animal$) and axioms (such as $Dog \sqsubseteq Animal$) are defined in specific home packages.

Definition 1 (Package): Let $O = (S, A)$ be an ontology, where S is the set of terms and A is the set of axioms over terms in S. A package $P = (\Delta_S, \Delta_A)$ of the ontology O is

a fragment of O, such that $\Delta_S \subseteq S$, $\Delta_A \subseteq A$. The set of all possible packages is denoted as Δ_P.

A term $t \in \Delta_S$ or an axiom $t \in \Delta_A$ is called a *member* of P, denoted as $t \in P$. P is called the *home package* of t, denoted as $\mathcal{HP}(t) = P$.

Terms can be names of classes (i.e., concepts), properties (i.e., roles), or instances (i.e., individuals). For example, for an ontology that states that tom (individual) is a Cat (concept) and a Cat (concept) $eats$ (role) $Mouse$ (concept), the terms of the ontology include $tom, Cat, eats, Mouse$.

We assume that each package, each term and each axiom has a unique identifier, such as a URI. For example, for a term t and package P, $t \in P$, both t and P are actually represented by some URIs.

A package can use terms defined in another package. In other words, an existing package can be reused by or imported into another package.

Definition 2 (Foreign Term and Importing): A term that appears in a package P, but has a different home package Q is called a *foreign term* in P. We say that P *imports* Q and we denote this as $P \mapsto Q$.

The importing closure $I_{\mapsto}(P)$ of a package P contains all packages that are directly or indirectly imported into P, where direct and indirect importing are defined as:

- (direct importing) $R \mapsto P \Rightarrow R \in I_{\mapsto}(P)$
- (indirect importing) $Q \mapsto R$ and $R \in I_{\mapsto}(P) \Rightarrow Q \in I_{\mapsto}(P)$

A Package-extended Description Logic ontology, or a **P-DL** ontology consists of multiple packages, each of them expressed in DL.

For example, the animal ontology O has two packages:

$\mathbf{P_{Animal}}$

Terms: $1:Dog, 1:Carnivore, 1:Animal, 1:eats$
Axioms:

(1a) $1:Dog \sqsubseteq 1:Carnivore$
(1b) $1:Carnivore \sqsubseteq 1:Animal$
(1c) $1:Carnivore \sqsubseteq \forall 1:eats.1:Animal$

$\mathbf{P_{Pet}}$

Terms: $2:PetDog, 2:Pet, 2:DogFood$
Foreign Terms: $1:Dog, 1:Animal, 1:eats \in P_{animal}$
Axioms:

(2a) $2:PetDog \sqsubseteq 1:Dog \sqcap 2:Pet$
(2b) $2:Pet \sqsubseteq 1:Animal$
(2c) $2:PetDog \sqsubseteq \exists 1:eats.2:DogFood$

We will omit the prefix "1:" and "2:" when there is no confusion. Here, $\mathbf{P_{Pet}}$ *imports* $\mathbf{P_{Animal}}$, since a term defined in $\mathbf{P_{Animal}}$ is referred in $\mathbf{P_{Pet}}$. The package domain in this example Δ_P is $\{P_{Animal}, \mathbf{P_{Pet}}\}$. $\mathbf{P_{Pet}}$ extends the ontology in $\mathbf{P_{Animal}}$ with assertions that a Dog may also be a Pet.

B. Package Hierarchy

Axioms in the packages of an ontology specify the term semantic structure of the ontology. However, real-world ontologies also call for fine-grained *organizational* structure due to several reasons (see also the previous section):

- For flexible partial reuse of an ontology;
- For organization of terms and axioms in a structure different from the semantic structure;
- For the management of an ontology in collaborative environments, where ontology modules are created and maintained by different people with different levels of privileges.

In our package-extended ontology, we introduce hierarchy as the organizational structure of the ontology. A package can be declared as a *sub package* of another package. Therefore, the complete ontology will have an organizational hierarchy, in addition to the semantic structure. Formally, we define package nesting as follows:

Definition 3 (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.

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

- $P_1 \in_N P_2 \rightarrow P_1 \in_N^* P_2$ (or short as $\in_N \rightarrow \in_N^*$)
- $P_1 \in_N^* P_2$ and $P_2 \in_N^* P_3 \rightarrow P_1 \in_N^* P_3$ (or short as $\in_N^* = (\in_N^*)^+$)

For example, we can declare that the *Pet* package is a sub package of the general *Animal* package: $P_{Pet} \in_N P_{Animal}$, and 'MyPet' package is a sub package of the 'Pet' package $P_{MyPet} \in_N P_{Pet}$, therefore $P_{MyPet} \in_N^* P_{Animal}$.

C. Scope Limitation Modifiers

In classical ontology languages such as OWL, all terms and axioms are globally visible and reusable. However, an open and collaborative environment of ontology construction requires selective limitation of term and axiom *scopes* for localized semantics, knowledge hiding and safe collaborative building of ontology.

Based on such considerations, we associate *scope limitation modifiers* (SLM) with terms and axioms defined in a package. An SLM controls the visibility of the corresponding term or axiom to entities on the web, in particular, to other packages. For example, a term with SLM 'public' can be visited from any package (see Fig. 3). Formally, a SLM is defined as follows:

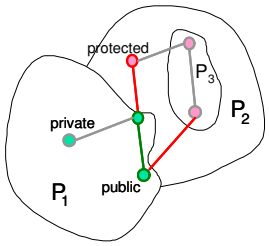

Fig. 3. Package-extended Ontology
The ontology has three packages $\mathbf{P_1}, \mathbf{P_2}, \mathbf{P_3}$. $\mathbf{P_3}$ is nested in $\mathbf{P_2}$. A public term in $\mathbf{P_1}$ is visible to $\mathbf{P_2}$, while a private term in $\mathbf{P_1}$ is only visible in the home package $\mathbf{P_1}$.

Definition 4 (SLM): The scope limitation modifier of a term or an axiom t_K in package K is a boolean function $f(p, t_K)$,

where p is a URI, the entity identified by p can access t_K iff $f(p,t) = \textbf{TRUE}$. We denote $t_K \in_f K$.

An entity on the web can be a user (e.g. emailto://baojie@cs.iastate.edu), a web program (e.g. http://www.foo.com/query.jsp), or another package (e.g. http://boole.cs.iastate.edu/animal/pet.powl)

In particular, we define three default SLMs as follows:

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

We can also define other types of SLMs as needed. For example, $\forall p, friend(p,t) := (p = P_1)$ will grant the access of t to a particular package P_1. An SLM can also be a complex function such as:

$$\forall p, f(p,t) := (p = \mathcal{HP}(t)) \vee (p \text{ LIKE } '*\text{CS.IASTATE.EDU}')$$

where LIKE is a string comparison operator. It guarantees the access of t to its home package and any entity in the cs.iastate.edu domain.

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

For example, the general animal package can be refined as:

$\mathbf{P_{Animal}}$

public:
terms: $Dog, Carnivore, Animal$
axioms:
$(1a)$ $Carnivore \sqsubseteq Animal$
$(1b)$ $Carnivore \sqsubseteq \forall eats.Animal$
protected:
terms: $Carnivora, CanisFamiliaris$
axioms:
$(1c)$ $Carnivora \equiv Carnivore$
$(1d)$ $Dog \equiv CanisFamiliaris$
private:
terms: $Canis$
axioms:
$(1e)$ $Canis \sqsubseteq Carnivora$
$(1f)$ $CanisFamiliaris \sqsubseteq Canis$

The scope limitation is designed to support both semantic encapsulation and knowledge hiding. The public terms and axioms represent high abstraction level knowledge, while the fine-grained knowledge is hidden. Furthermore, since the knowledge about $Canis$ is not intended to be further refined by any sub packages (such as the $\mathbf{P_{Pet}}$ package), it is available only locally.

A scope limitation modifier has several features:

- It aims at *partial hiding* of semantics. The hiding is partial not only because there is only a selected part of a module that is hidden, but also because the hidden semantics may still be used in inference, as long as the hidden part is not exposed. For example, $Canis \sqsubseteq Carnivore$ is hidden, but a user may be able to infer indirectly that $Dog \sqsubseteq Animal$;
- It enables ontology *polymorphism*, in the sense that one ontology can be browsed and queried differently from different points of view. For example, the same 'Animal' ontology exposes different sets of terms and axioms to the public domain and to the private domain (e.g., paid users). Therefore, the same ontology can be partially reused in different ways by different people.

The aforementioned definitions of package-extended ontology are summarized in Table I. Here, Δ_S is the ontology term domain, which is the set of all possible names in the ontology; Δ_A is the set of all possible axiom identifiers; Δ_P is the domain of all possible packages.

IV. Semantics of Package-extended Ontology

In the previous section, we have defined the language elements of package-extended ontology. This section will further investigate the semantics of package-extended ontology, in particular, P-DL.

A. Local Interpretation

DL languages have model theoretical interpretation [1] for their semantics:

Definition 5 (Interpretation): An interpretation of a description logic is a pair $\mathcal{I} =< \triangle^{\mathcal{I}}, (.)^{\mathcal{I}} >$, where $\triangle^{\mathcal{I}}$ contains a nonempty set of objects and $(.)^{\mathcal{I}}$ is a function that maps each class name C to $C^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}}$; each role name P to $P^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$, and each instance name i to $i^{\mathcal{I}} \in \triangle^{\mathcal{I}}$.

In other words, an interpretation of a DL ontology constructs a world consisting of a set of objects, and maps each term in the ontology into an object (individual), a set of objects (concept) or a binary relation of objects (role). For example, consider a very simple description logic in which there are only concept inclusions and atomic roles, as below:

$Dog \sqsubseteq Animal$
role: $eats$
individual: $Dog(goofy)$

This description logic ontology could have an interpretation $\mathcal{I}$, such that $\Delta^{\mathcal{I}} = \{a, b\}$, $(.)^{\mathcal{I}}$ maps concept Dog to $Dog^{\mathcal{I}} = \{a\} \subseteq \Delta^{\mathcal{I}}$, $Animal$ to $Animal^{\mathcal{I}} = \{a, b\} \subseteq \Delta^{\mathcal{I}}$, role $eats$ to $eats^{\mathcal{I}} = \{(a, b)\} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and individual $goofy$ to $goofy^{\mathcal{I}} = a \in \Delta^{\mathcal{I}}$. In this interpretation, $Dog^{\mathcal{I}} \subseteq Animal^{\mathcal{I}}$, therefore the inclusion axiom $Dog \sqsubseteq Animal$ in the ontology is satisfied.

For each package in a P-DL, we can define the local interpretation of the package.

Definition 6 (Local Interpretation): A local interpretation of a package P is a pair $\mathcal{I}_P =< \Delta^{\mathcal{I}_P}, (.)^{\mathcal{I}_P} >$, where $\triangle^{\mathcal{I}_P}$ the set of all objects and $(.)^{\mathcal{I}}_P$ is a function that maps each concept name C to $C^{\mathcal{I}_P} \subseteq \Delta^{\mathcal{I}_P}$; each role name R to $R^{\mathcal{I}_P} \subseteq \Delta^{\mathcal{I}_P} \times \triangle^{\mathcal{I}_P}$, and each individual name i to $i^{\mathcal{I}_P} \in \Delta^{\mathcal{I}_P}$.

Note that in this definition, the term (concept, role or individual) name can be either defined in P or is a foreign term

TABLE I

SYNTAX AND SEMANTICS OF PACKAGE-EXTENDED ONTOLOGY

Constructor	Syntax	Semantics
package	P	$P \in \Delta_P$
membership	$t \in P$ or $member(t, P)$	$member \subseteq (\Delta_S \cup \Delta_A) \times \Delta_P$
home package	$\mathcal{HP}(t)$	$\mathcal{HP}(t) = P$, where $t \in P$
nesting	$\in_N$	$\in_N \in \Delta_P \times \Delta_P$
transitive nesting	$\in_N^*$	$\in_N \rightarrow \in_N^*$, $\in_N^* = (\in_N^*)^+$
SLM	SLM(p, t)	p can access $t \in (\Delta_S \cup \Delta_A)$ iff SLM(p,t)=**TRUE**
	public(p, t)	$\forall p, public(p, t) :=$ **TRUE**
	private(p, t)	$\forall p, private(p, t) := (t \in p)$
	protected(p, t)	$\forall p, protected(p, t) := (t \in p)$ or $(p \in_N^* \mathcal{HP}(t))$

defined in other packages. For example, given the 'Animal' ontology (page 4) with packages $\mathbf{P_{Animal}}$ and $\mathbf{P_{Pet}}$, we have possible local interpretations for the two packages as in Fig. 4 (a) and (b). Foreign terms are represented by dot lines in the figure.

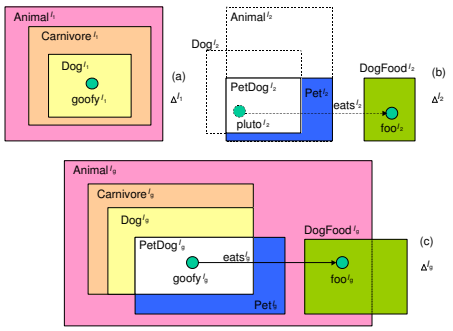

Fig. 4. Local and Global Interpretation of the Animal Ontology
(a) $\mathcal{I}_1$ is a local interpretation of $\mathbf{P_{Animal}}$;
(b) $\mathcal{I}_2$ is a local interpretation of $\mathbf{P_{Pet}}$;
(c) $\mathcal{I}_g$ is a global interpretation of the ontology consisting of the two packages.

The two local interpretations have noticeable characteristics:

- Since a local interpretation explains everything in the local domain, the semantics of foreign terms is not 'imported' into the local domain. Instead, they are explained the same as a local term. For example, Dog and $Animal$ have interpretations in both $\mathcal{I}_1$ and $\mathcal{I}_2$. While in the domain of $\mathcal{I}_1$ ($\mathbf{P_{Animal}}$), $Dog^{\mathcal{I}_1} \subseteq Animal^{\mathcal{I}_1}$ must be true, it is not required from the local point of view of $\mathcal{I}_2$ ($\mathbf{P_{Pet}}$). Therefore, the semantics of $\mathbf{P_{Animal}}$ is not imported into $\mathbf{P_{Pet}}$ for *local* interpretations.
- The same term can be interpreted differently in two packages. For example, the two interpretations $Dog^{\mathcal{I}_1} = \{goofy\}$ and $Dog^{\mathcal{I}_2} = \{pluto\}$ are not necessarily identical; they may be different on the individual names or even numbers of individuals.

We assume a package always has a local interpretation, since local consistency is a natural and necessary requirement for web ontologies. Indeed, if local consistency cannot be guaranteed, integrity of any information that is based on the package cannot be guaranteed, making such a package useless. On the other hand, global consistency is a much stronger requirement, and in practice, cannot be guaranteed given the fact that individual ontology modules may be developed by independent groups. For example, if the package $\mathbf{P_{Pet}}$ has two more axioms $DogFood \sqsubseteq CannedFood$ and $CannedFood \sqsubseteq \neg Animal$, no possible interpretation exists for the global ontology obtained by combining the two packages.

B. Global Interpretation

A global interpretation is a possible interpretation for all packages in an ontology. It interprets the ontology not from the *local* point of view of a *single* package, but from the *global* point of view of *all* packages. Formally, the global interpretation of an ontology is defined as follows:

Definition 7 (Global Interpretation): A global interpretation of a set of packages $\{P_i\}$ with local interpretations $\mathcal{I}_i = \langle \Delta^{\mathcal{I}_i}, (.)^{\mathcal{I}_i} \rangle$, $i = 1, \cdots m$ is $\mathcal{I}_g = \langle \Delta^{\mathcal{I}_g}, (.)^{\mathcal{I}_g} \rangle$, where $\Delta^{\mathcal{I}_g} = \cup_{i=1}^{m} \Delta^{\mathcal{I}_i}$ and $(.)^{\mathcal{I}_d}$ maps each concept name C to $C^{\mathcal{I}_g} \subseteq \Delta^{\mathcal{I}_g}$; each role name R to $R^{\mathcal{I}_g} \subseteq \Delta^{\mathcal{I}_g} \times \Delta^{\mathcal{I}_g}$, and each individual name i to $i^{\mathcal{I}_g} \in \Delta^{\mathcal{I}_g}$.

Each $\mathcal{I}_i$ is called a *projection* of $\mathcal{I}_g$. We have: (1) $\Delta^{\mathcal{I}_i} \subseteq \Delta^{\mathcal{I}_g}$; and (2) for each concept or role name t, $t^{\mathcal{I}_i} \subseteq t^{\mathcal{I}_g}$ and for each individual name t, $t^{\mathcal{I}_i} = t^{\mathcal{I}_g}$. Such a relation is denoted as $(.)^{\mathcal{I}_i} \subseteq (.)^{\mathcal{I}_g}$.

For example, a possible global interpretation for the 'Animal' ontology is given in Fig. 4 (c). The package-extended ontology is consistent if and only if a global interpretation exists. The global interpretation is consistent with the semantics from *all* packages that are used. When a global interpretation exists, it corresponds to the 'global' point of view of the combined ontology and each local interpretation is a proper subset of it. For example, if we define $goofy^{\mathcal{I}_1} = pluto^{\mathcal{I}_2} = goofy^{\mathcal{I}_g}$, $foo^{\mathcal{I}_2} = foo^{\mathcal{I}_g}$, we have $\Delta^{\mathcal{I}_1} \subseteq \Delta^{\mathcal{I}_g}$, $(.)^{\mathcal{I}_1} \subseteq (.)^{\mathcal{I}_g}$ and $\Delta^{\mathcal{I}_2} \subseteq \Delta^{\mathcal{I}_g}$, $(.)^{\mathcal{I}_2} \subseteq (.)^{\mathcal{I}_g}$.

C. Distributed Interpretation

While all packages can share the same global interpretation, they may also have different 'distributed' interpretations, which are integrated but local points of view over a set of ontology modules. For example, consider the package $\mathbf{P_{Livestock}}$ in the 'Animal' ontology containing the following knowledge: *a domestic animal is an animal and it will never eat another domestic animal, livestock is a type of domestic animal, dog and horse are types of livestock.*

$\mathbf{P_{Livestock}}$

Terms: $DomesticAnimal, Livestock, Horse$
Foreign Terms: $Dog, Animal, eats \in P_{animal}$

Axioms:

$DomesticAnimal \sqsubseteq Animal$
$DomesticAnimal \sqsubseteq \forall eats.\neg DomesticAnimal$
$Livestock \sqsubseteq DomesticAnimal$
$Dog \sqcup Horse \sqsubseteq Livestock$

Although the three packages ($\mathbf{P_{Animal}}$, $\mathbf{P_{Pet}}$, $\mathbf{P_{Livestock}}$) are consistent and a global interpretation exists, they still represent different views on the domain. For example, the satisfiability query $Dog \sqcap \exists eats.Dog$ (if a dog may possibly eat another dog) will be answered YES in $\mathbf{P_{Pet}}$, but NO in $\mathbf{P_{Livestock}}$. Formally, a distributed interpretation is defined as follows:

Definition 8 (Distributed Interpretation): A distributed interpretation of a set of packages $\{P_i|i = 1, ..., m\}$ witnessed from one package $P_k, 1 \leq k \leq m$, is a pair $\mathcal{I}_d = \langle \Delta^{\mathcal{I}_d}, (.)^{\mathcal{I}_d} \rangle$, where $\Delta^{\mathcal{I}_d}$ is the set of all possible individuals, and for each term in $(P_k \cup I_{\mapsto}(P_k)) \cap (\{P_i|i = 1, ..., m\})$, $(.)^{\mathcal{I}_d}$ maps each concept name C to $C^{\mathcal{I}_d} \subseteq \Delta^{\mathcal{I}_d}$; each role name R to $R^{\mathcal{I}_d} \subseteq \Delta^{\mathcal{I}_d} \times \Delta^{\mathcal{I}_d}$, and each individual name i to $i^{\mathcal{I}_d} \in \Delta^{\mathcal{I}_d}$. P_k is called the *witness* of $\mathcal{I}_d$.

Thus, a distributed interpretation witnessed by a package P represents the semantics of axioms in P and axioms in all of its recursively imported packages. Note that:

- It is different from a *local interpretation* of P in that the later only represents semantics of local axioms in P and all foreign term are treated only as symbols.
- It is also different from a *global interpretation* of the whole package-extended ontology in that the later is a model for the integrated knowledge base of *all* packages, while a distributed interpretation is a model for *some* (not necessarily all) packages in the ontology. For a certain package, a distributed interpretation may exist even when the whole ontology has no global interpretation.

The intuition behind the distinction among the three interpretations can be illustrated using the US legislature system:

1) Each specific act of state legislature has its own set of rules representing the point of view (local interpretation) of the state legislature, e.g., *speed limit is 65mph*.
2) The laws of each state, when taken together with the applicable federal laws, represent a distributed interpretation for that particular state, and this interpretation may be different from those of other states (because of possible differences between state laws, e.g., *differences with respect to the speed limit*).
3) The consensus (the global interpretation) of laws of all states may not exist, since different states may have incompatible attitudes on the same issue (e.g., *death penalty*).

V. Case Study

/ We revise the well-known wine ontology (Fig. 5) into P-DL to exhibit features of package-extended description logics. The original wine ontology is given in two OWL/RDF files[1]

[1] http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine.rdf and http://www.w3.org/TR/2004/REC-owl-guide-20040210/food.rdf

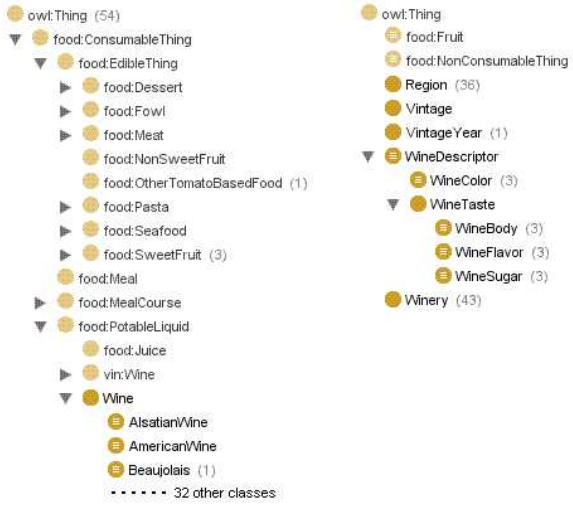

Fig. 5. Wine Ontology(Part)

focused on wine knowledge and general food knowledge, respectively. However, such division into different files, a.k.a., XML name spaces, is not a satisfactory solution to semantically sound and efficient modularization of the ontology, as the original wine ontology presents the following problems:

- It cannot be partially reused. The two modules mutually import each other, therefore a user has to import ALL the wine and food modules, although only a small part of the ontology may be needed.
- The ontology is not well-organized. For example, there is knowledge about region defined in wine.rdf, which is not specific only to represent wine knowledge, but is rather general and may also be reused in other applications.
- The ontology organization is not suitable for collaborative building. For example, a wine expert from France and another expert from United States cannot work on wine.rdf *concurrently* and safely (i.e., there is no principled way to avoid unwanted coupling).

We will show how the wine ontology can be transformed into a P-DL ontology in order to overcome these limitations.

A. Modularization

The P-DL version of the ontology consists of several packages (Fig. 6). Each package contains a set of terms (concepts, roles, and individuals) and associated axioms. For example, the **Winery** package contains classes, such as, $Vintage, Winery, WineDescription, VintageYear$, and their subclasses, individuals of those classes, and some properties, such as $hasMarker$ or $hasWineDesctiptor$. Terms in one package may have no asserted semantic relations (e.g., classes $Winery$ and $Vintage$), but are related to describe one aspect of the domain of interest (here is wine).

The decomposition is meant to:

- **Reduce the importing cost**. In the original OWL/RDF ontology, there is mutual importing between the two files, since both ontology modules use some terms defined

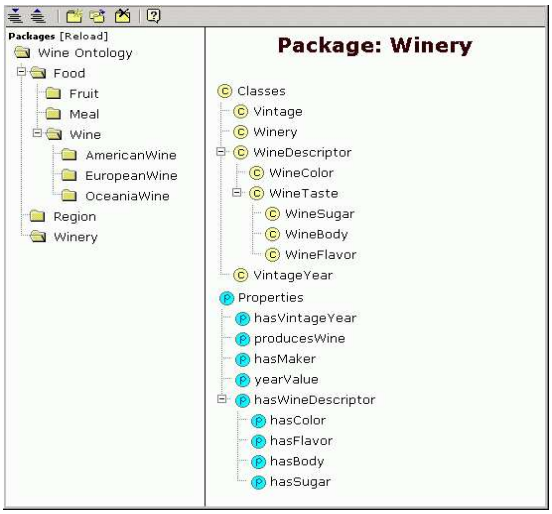

Fig. 6. Package-extended Version of Wine Ontology: the Package Hierarchy and the Content of the Package 'Winery'

in the other module. Therefore, reusing only 'Food' ontology module is not possible. However, it turns out that only a few terms defined in wine.rdf are used in food.rdf. These terms can be organized in a smaller separate package (**Winery**, see Fig. 6) to avoid importing unneeded chunks of wine knowledge into the **Food** module. For the same reason, **Food** package has sub packages **Fruit** and **Meal**, since only fruit knowledge is needed in the $Wine$ module, while the other part of the food knowledge is not related to wine. Such fine-grained importing reduces communication, parsing and reasoning costs, when reusing the food module. It also reduces the risk of unwanted coupling and semantic clashes.

- **Improve partial reusability**. Some parts of the ontology, such as region knowledge, can be reused in other applications. However, lack of organizational separation of the original OWL/RDF ontology makes it hard for such partial reuse. In the P-DL version of the wine ontology, the package **Region** is separated to enable partial reuse of the ontology, if a user only needs this part of knowledge.
- **Ensure better readability and collaborative building**. Large concept taxonomies can be divided into smaller modules, even when the semantic structure is not modularized. For example, in the original OWL/RDF ontology, there are a large number of subclasses directly under the concept $Wine$. We can *organize* them in a package hierarchy, such as package **Wine**, **AmericanWine** and **EuropeanWine**, even when the semantic hierarchy has no such direct division. Therefore, we can manipulate the ontology with smaller, but more focused modules. This is beneficial for both collaborative building of the ontology (e.g., wine experts from U.S. and France work on different packages) and better readability and understandability of the ontology.

B. Knowledge Hiding

Scope limitation modifiers can be used in the wine ontology for multiple purposes:

- **Hiding critical information**. For example, the **Meal** package contains detailed recipes for meal courses, e.g.
  $a) DessertCourse \sqsubseteq MealCourse$
  $b) DessertCourse \equiv MealCourse \sqcap \forall hasFood.Dessert$
  $c) DessertCourse \sqsubseteq \forall hasDrink.(\exists hasBody.Full)$
  $d) DessertCourse \sqsubseteq \forall hasDrink.(\exists hasFlavor.Strong)$
  $e) DessertCourse \sqsubseteq \forall hasDrink.(\exists hasSugar.Sweet)$
  The provider of the ontology may wish to hide such details for commercial purpose; accordingly, the provider will assign an SLM 'public' to the axiom $a)$ and SLMs 'private' to the axioms $b) - e)$.
- **Semantic Encapsulation**. For example, the **Wine** package contains individuals of $WineGrape$ (e.g., $CabernetFranceGrape$), which are only intended to define wine types, but are of no interest to users. Therefore, those individuals can be declared as protected, which means that only package **Wine** and its decedent packages can use them. They are hidden from user access, being seen as the implementation details of the package.
- **Safe collaborative ontology design**. The use of SLMs in the design stage and the released ontology 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, the terms in the package **Food**, except for classes $Fruit, Meal, MealCourse, Wine$, can be made private terms in the design stage, since those terms are not intended to be extended by any other packages. Hiding of those term will minimize the risk of unwanted semantic coupling. We can change them to public terms in the released ontology.

VI. Related Work

A. Distributed Description Logics

Several distributed logic systems have been studied during the recent years. Examples include Local Model Semantics [14] and Distributed First Order Logic (DFOL) [15], which emphasize local semantics and the compatibility relations among local models.

Inspired by DFOL, Borgida and Serafini [6] extend 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 "backflow" of information among modules connected by a bridge rule.

Contextual ontology framework, a formalism based on DDL, emphasizes localized semantics in ontologies. Contextual ontology keeps the content local and maps the content to other ontologies via explicit bridge rules. Bouquet *et al.* [7]

proposed CTXML, which includes a hierarchy-based ontology description and a context mapping syntax. They further combined CTXML and OWL into Context OWL (C-OWL) [8] , a syntax for bridge rules over OWL ontology.

Serafini and Tamilin [24], [25] define a sound and complete distributed tableau-based reasoning procedure, which is built as an extension to standard DL tableau. In [25], they also describe the design and implementation principles of a distributed reasoning system, called DRAGO (Distributed Reasoning Architecture for a Galaxy of Ontologies), that implements such a distributed decision procedure. Serafini *et al.* [23] further define a fix-point semantics for bridge rules.

DDL and P-DL share some similarities:

- A global ontology is composed of multiple modules in both formalisms.
- There is no imposed universal global semantic. Instead, local semantics are expresed with respect to local points of view.
- Semantic connections between modules are always directional.

However, there are also several differences between DDL and P-DL:

- P-DL is more expressive than DDL. Bridge rules, such as $\overset{\sqsubseteq}{\leftarrow}$(INTO) and $\overset{\sqsupseteq}{\rightarrow}$(ONTO) connect only *atomic* concepts. For example, the DDL version of the animal ontology (see page 4) will have a bridge rule $1 : Dog \overset{\sqsupseteq}{\rightarrow} 2 : PetDog$. However, if a relation involves both role names and concept names from different ontology modules, e.g. $2 : PetDog \sqsubseteq \exists 1 : eats.2 : DogFood$, this can not be expressed with bridge rules.
- P-DL and DDL differ in the way they handle inconsistencies among ontology modules. DDL allows local ontologies to be internally inconsistent and focuses on preventing propagation of local inconsistencies. If a module is locally inconsistent, the entire module is sacrificed (discarded). On the other hand, our approach assumes that each module is locally consistent, and discards only those axioms that are inconsistent with other modules, in order to construct a consistent ontology. Since the ontology modules are usually autonomous, it is more natural (and easier) to ensure local consistency.
- DDL and P-DL differ in how they handle imported semantics. In DDL, directed semantic importing is done by bridge rules, therefore there are no foreign terms in an ontology. In P-DL, a foreign term can be directly used by a module.
- DDL emphasizes connecting existing ontologies, where existing modules are articulated with semantic mapping. On the other hand, P-DL emphasizes collaborative ontology construction, wherein new modules are extended from existing modules.
- P-DL further introduces knowledge hiding for localized semantics, which is missing in DDL.

B. Other Modular Ontology Proposals

Calvanese *et al.* [9] proposed a view-based query answering mechanism for ontology integration. An Ontology Integration System (OIS) is a triple $\langle G, S, M_{G,S} \rangle$, where G is the global ontology, S is the set of local ontologies and $M_{G,S}$ is the mapping between G and the local ontologies in S. The mapping can be global-to-local or local-to-global. However, the assumption of the existence of a global ontology is too strong for real world Semantic Web applications, where no global semantics can be guaranteed across independent, semantically heterogeneous and autonomous information sources.

The *Modular Ontology* [26] offers a way to exploit modularity in reasoning. It defines an architecture that supports local reasoning by compiling implied subsumption relations. It also provides 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.

Grau *et al.* [19] explore using $\mathcal{E}$-connections to extend OWL or $\mathcal{SHIQ}$ and this approach is straightforward to implement on existing tableau OWL reasoners. $\mathcal{E}$-connections are more expressive than DDL bridge rules in that a role can have domain and range from different modules. P-DL is strictly more expressive than $\mathcal{E}$-connections in that it supports other role and nominal constructions such as role inclusions between modules.

Serafini and Wache [21] give a survey of existing ontology mapping languages, such as DDL, C-OWL, OIS, DLII [10], and $\mathcal{E}$-connections, by translating them into distributed first order logic. None of those approaches provides scope limitation, therefore they are limited on ontology partial reuse and avoidance of unintended coupling.

C. Knowledge Hiding in Ontology

Fikes *et al.* [13] mentioned integration of modular ontologies in the Ontolingua system and restricting 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 idea of symbol access restriction, but it is more flexible and expressive.

Efforts aimed at developing formal languages to control ontology access scope include Extensible Access Control Markup Language (XACML) [17] and policy languages [11], [27]. Giereth [16] studied hiding part of RDF, where sensitive data in an RDF-graph is encrypted for a set of recipients, while all non-sensitive data remain publicly readable. However, those efforts are aimed at safe access on language or syntactic level. On the other hand, SLM in P-DL aims at knowledge hiding on semantic level, where the hiding is not *total*, but *partial*, i.e., hiding semantics can still be used in safe indirect inferences.

Farkas [12] studied unwanted inferences problem in semantic web data on XML, RDF or OWL level. Our approach to SLM and concealable reasoning is a more principled formalism to avoid unwanted inferences and with better defined localized semantics.

VII. Summary and Discussions

In this paper, we have explored package-extended description logics and its semantics. The major contributions of this paper include:

- The introduction of package-extended ontologies as a framework for collaborative ontology construction, sharing, and use.
- The introduction of scope limitation of ontology terms to support partial knowledge hiding in knowledge sharing. SLMs help avoid unintended coupling between parts of ontologies, help ensure prevention of unintended disclosure of hidden knowledge, and support semantic encapsulation.
- The formal semantics of package-extended ontology and the differentiation between local interpretation, global interpretation and distributed interpretation.

The next step of this work will be focused on the problem of reasoning in P-DL, which differs from the standard reasoning problem in DL on the following aspects:

- It is a *distributed* reasoning process, instead of a centralized process. The global reasoning process is built upon local reasoning services, offered by each of the individual packages.
- It needs to contend with possible *inconsistencies* among different ontology modules, instead of always assuming a single consistent ontology.
- It needs to respect the *privacy* of ontology modules. If a package only exposes a part of its knowledge base, the other ontology packages should be prevented from reconstructing the hidden knowledge by using the reasoning service supported by the package in question.

Work in progress also includes the improvement to existing tools to edit package-extended ontologies, such as Wiki@nt [2] and INDUS DAG-Editor [4].

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

References

[1] F. Baader and W. Nutt. Basic description logics. In Franz Baader, Diego Calvanese, and Deborah McGuinness et.al., editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 43–95. Cambridge University Press, 2003.

[2] Jie Bao and Vasant Honavar. Collaborative ontology building with wiki@nt - a multi-agent based ontology building environment. In *Proc. of 3rd International Workshop on Evaluation of Ontology-based Tools, located at the 3rd International Semantic Web Conference ISWC 2004, 8th November 2004, Hiroshima, Japan*, 2004.

[3] Jie. Bao and Vasant. 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.

[4] Jie Bao and Vasant Honavar. Collaborative package-based ontology building and usage. In *IEEE Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources, in ICDM2005*, 2005.

[5] T Berners-Lee, J Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001.

[6] Alexander Borgida and Luciano Serafini. Distributed description logics: Directed domain correspondences in federated information sources. In *CoopIS/DOA/ODBASE*, pages 36–53, 2002.

[7] P. Bouquet, A. Dona, L. Serafini, and S. Zanobini. Conceptualized local ontologies specification via ctxml. In *AAAI-02 Workshop on Meaning Negotiation (MeaN-02) July 28, 2002, Edmonton, Alberta, Canada*, 2002.

[8] P. Bouquet, F. Giunchiglia, and etc. 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.

[9] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. A framework for ontology integration. In *The Emerging Semantic Web*, 2001.

[10] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Description logics for information integration. In *Computational Logic: Logic Programming and Beyond*, pages 41–60, 2002.

[11] Ernesto Damiani, Sabrina De Capitani di Vimercati, Cristiano Fugazza, and Pierangela Samarati. Extending policy languages to the semantic web. In *ICWE*, pages 330–343, 2004.

[12] C. Farkas. *Web and Information Security*, chapter Data Confidentiality on The Semantic Web: Is There an Inference Problem? Chapter IV, pages 73–91. Idea Group Inc, 2006.

[13] Richard Fikes, Adam Farquhar, and James Rice. Tools for assembling modular ontologies in ontolingua. In *AAAI/IAAI*, pages 436–441, 1997.

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

[15] 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.

[16] Mark Giereth. On partial encryption of rdf-graphs. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *International Semantic Web Conference*, volume 3729 of *Lecture Notes in Computer Science*, pages 308–322. Springer, 2005.

[17] S. Godik and T. Moses. Oasis extensible access control markup language (xacml). OASIS Committee Secification cs-xacml-specification-1.0, November 2002, 2002.

[18] Asun Gomez-Perez, Juergen Angele, Mariano Fernandez-Lopez, V. Christophides, Athur Stutt, and York Sure. Deliverable 1.3: A survey on ontology tools, 2002.

[19] Bernardo Cuenca Grau, Bijan Parsia, and Evren Sirin. Working with multiple ontologies on the semantic web. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 620–634. Springer, 2004.

[20] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1), 2003.

[21] Heiner Stickenschmidt Luciano Serafini and Holger Wache. A formal investigation of mapping language for terminological knowledge. In *19th IJCAI*, 2005.

[22] G. Schreiber and M. Dean. Owl web ontology language reference, February 2004.

[23] Luciano Serafini, Alexander Borgida, and Andrei Tamilin. Aspects of distributed and modular ontology reasoning. In *IJCAI*, pages 570–575, 2005.

[24] Luciano Serafini and Andrei Tamilin. Local tableaux for reasoning in distributed description logics. In *Description Logics*, 2004.

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

[26] Heiner Stuckenschmidt and Michel Klein. Modularization of ontologies - wonderweb: Ontology infrastructure for the semantic web, 2003.

[27] Gianluca Tonti, Jeffrey M. Bradshaw, Renia Jeffers, Rebecca Montanari, Niranjan Suri, and Andrzej Uszok. Semantic web languages for policy representation and reasoning: A comparison of kaos, rei, and ponder. In *International Semantic Web Conference*, pages 419–437, 2003.