An Interface-based Ontology Modularization Framework for Knowledge Encapsulation

Faezeh Ensan and Weichang Du

Faculty of Computer Science,
University of New Brunswick, Fredericton, Canada
{faezeh.ensan,wdu}@unb.ca

Abstract. In this paper, we present a framework for developing ontologies in a modular manner, which is based on the notions of interfaces and knowledge encapsulation. Within the context of this framework, an ontology can be defined and developed as a set of ontology modules that can access the knowledge bases of the others through their well-defined interfaces. An important implication of the proposed framework is that ontology modules can be developed completely independent of each others' signature and language. Such modules are free to only utilize the required knowledge segments of the others. We describe the interface-based modular ontology formalism, which theoretically supports this framework and present its distinctive features compared to the exiting modular ontology formalisms. We also describe the real-world design and implementation of the framework for creating modular ontologies by extending OWL-DL and modifying the Swoop interfaces and reasoners.

1 Introduction

OWL-DL has been well established and widely used in the recent years as an expressive description logic based language for representing ontologies. Nonetheless, several challenges still exist in efficiently creating large-scale OWL-DL ontologies specially for complex domains. Developing a large monolithic ontology can lead to performance difficulties in reasoning, management challenges when some parts of the ontology changes based on new domain requirements, and also issues in ontology integration when several parts of an ontology have been developed by different groups of experts.

Recently, the development of ontologies in a modular manner has been proposed to address the above mentioned issues [19]. The idea of modularization can also be seen in the software engineering field and mainly in Object Oriented design where complex software systems are modeled as a set of self-contained components [18]. The behavior of these components are defined through their interfaces which are separated from their later detailed implementation. Consequently, components can utilize each others' functions without being consciously aware of each other detailed implementation. Then implementation changes can occur even after logical component inter-connections have been specified.

Due to their perceived advantages, a considerable amount of work has been dedicated to creating new formalisms for ontologies which support developing ontologies in a modular manner. Distributed Description Logics (DDL) [3], Package Based Description Logics P-DL) [1], E-connections [14] and Semantic Import [2] are among such formalisms. DDL defines a modular ontology as a set of ontology modules which are connected through ‘bridge rules’. A bridge rule forms a mapping between two concepts of two different ontology modules. Further, Context OWL (C-OWL) [4] has been introduced as an extension to OWL to support the syntax and semantics of such bridge rules. E-connections is another formalism, which supports ontology modularization by introducing a new type of roles (called links) whose domain and range belong to different ontology modules. The authors in [10] describe their extension to OWL for supporting E-connections ‘links’ and also the modification of the Pellet reasoning engine to support its semantics. The Semantic Import and P-DL formalisms allow ontology modules to import the knowledge base elements of each other.

Despite these numerous efforts, it seems that ontology modularization is still far from the level of maturity needed to be accepted as an established method for developing ontologies. For instance, E-connections is based on a limiting precondition that the domain of the ontology modules need to be completely disjoint. DDL restricts mappings to ontology concepts and does not support role mappings or creating complex concepts using foreign terms. The decidability of consistency checking in P-DL in its current form can only be proven when it restricts importing terms to concepts and also when all of the component modules are in the same description logics language [1]. Moreover, as it is shown in [9] P-DL consists of some ambiguities in its introduced semantics such that for performing a reasoning task on a module, the union of the knowledge bases of all modules should be processed. Semantic Import provides reasonable expressiveness that allows a module to use the others’ terms in its complex concepts, however, the properties of the formalization is being discussed under the assumption that a module imports all of the symbols of the others [2].

In addition to the technical issues discussed above, there is a missing feature in the existing formalisms, that is support for ‘knowledge encapsulation’, which would benefit ontology modularization. By knowledge encapsulation we mean providing support for ontology modules to define their main contents using well-defined interfaces, such that their knowledge bases can only be accessed by other modules through these interfaces. The advantages of knowledge encapsulation in ontology design and development can be enumerated as follows:

- Since ontology modules are connected indirectly through their interfaces, they can evolve independent of each others’ signature and knowledge bases. While the interfaces of a module do not change, its entire knowledge base can change without requiring other connected modules to change their signatures.
- An ontology module can express its knowledge content through different interfaces with different levels of complexity and completeness. Hence, mod-

ules can access those parts of an ontology module they need without being required to go through the complicated knowledge base.

– Using interfaces, the specification of the knowledge of an ontology module and the exact meaning of these content can be separated. Consequently, an ontology module can provide new meaning for a concept which is used by other modules through its interfaces.

In this paper, we propose a new framework for developing ontologies in a modular manner based on an interfaces-based formalism. This framework supports a type of knowledge encapsulation that allows ontology modules to define various interfaces through which other ontology modules can access their knowledge base. Based on this interface-based formalism, a modular ontology is defined as a set of ontology modules, which can be developed independent of each others' language and signature. Furthermore, the formalism addresses the technical issues existing in the current ontology modularization proposals. The interface-based formalism enjoys a great expressiveness power, which allows a module to create its knowledge base from the other modules' knowledge expressed through their interfaces. At the same time, it allows for partial reuse, i.e., it lets a module use only the necessary parts of the knowledge base of the other modules. In addition, the consistency checking of a modular ontology is decidable even though the modules are in different description languages.

The rest of this paper is organized as follow: Section 2 provides preliminaries regarding basics of description logics and also epistemic queries. Section 3 introduces the syntax and semantics of the proposed interface-based formalism. Section 4 presents the important features of the formalism. Section 5 illustrate the implementation of the framework for OWL ontologies. Section 6 discusses related works and finally, section 7 concludes the paper.

2 Preliminaries

OWL-DL provides an expressiveness equivalent to the $\mathcal{SHOIN}(D)$ Description Logic (DL). A DL knowledge base is defined as $\Psi = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ denotes TBox and comprises of a set of general inclusion axioms and A stands for ABox and comprise of a set of instance assertions. The signature of an ontology is defined as a set of all concept names (C_N), role names (R_N) and individulas (I_N) which are included its knowledge base. The semantic of a DL is defined by an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is a non-empty set of individuals and $\cdot^{\mathcal{I}}$ is a function which maps each $C \in C_N$ to $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, each $R \in R_N$ to $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and each $a \in I_N$ to an $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. An interpretation $\mathcal{I}$ satisfies a TBox axiom $C \sqsubseteq D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, satisfies an ABox assertion $C(a)$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and an ABox assertion $R(x, y)$ iff $\langle x^{\mathcal{I}}, y^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$. An interpretation $\mathcal{I}$ is a model of a knowledge base Ψ if it satisfies every TBox axiom and ABox assertion of Ψ. A knowledge base is consistent iff it has a model. A concept C is *satisfiable* if there is a model $\mathcal{I}$ for Ψ such that $C^{\mathcal{I}} \neq \emptyset$.

DL ontologies are based an open-world assumption, i.e., the knowledge base of an ontology is not considered to be complete and consequently if something cannot be proven, it cannot be assumed to be false based on the knowledge base. Nevertheless, there have been some proposals in the literature that attempt to augment the semantics of DLs with close-world reasoning capabilities. Epistemic operator K is introduced in [7] and allows queries whose result can be captured by the closed-world assumption approach. K queries ask about the facts that are known to be true with the extent of information available in the current knowledge of a given knowledge base. [5] investigates mechanisms for posing K epistemic queries to expressive DL knowledge bases. [17] shows the capability of the Pellet reasoning engine for answering K queries for concepts and roles that are posed to simple knowledge bases.

To have a formal understanding of K queries, let C be a concept in a description logic knowledge base Ψ, KC reports a set of individuals which are known to belong to C in every model of Ψ. An *epistemic interpretation* for Ψ is defined as $\mathfrak{I} = (\mathcal{J}, \mathcal{M})$, where $\mathcal{J}$ is an interpretation of Ψ with the domain Δ, and $\mathcal{M}$ is a set of interpretations for Ψ over Δ. The epistemic interpretation for simple epistemic concepts and roles are defined as below:
$(\mathrm{K}C)^{\mathfrak{I}} = \bigcap_{j \in \mathcal{M}} (C)^j$
$(\mathrm{K}R)^{\mathfrak{I}} = \bigcap_{j \in \mathcal{M}} (R)^j$
An epistemic model for a knowledge base Ψ is a maximal non-empty set $\mathcal{M}$ such that for every $\mathcal{J} \in \mathcal{M}$, $(\mathcal{J}, \mathcal{M})$ satisfies all TBox inclusion axioms and ABox assertions of Ψ. Consider an epistemic query $\mathrm{K}C(x)$ posed to a knowledge base Ψ, $\Psi \models \mathrm{K}C(x)$ if for every epistemic model $\mathfrak{I} = (\mathcal{J}, \mathcal{M})$ for Ψ, $x \in \mathrm{K}C^{\mathfrak{I}}$. An epistemic query $\mathrm{K}R(x, y)$ is also defined in the same way as $\Psi \models \mathrm{K}R(x, y)$ if for every epistemic model $\mathfrak{I} = (\mathcal{J}, \mathcal{M})$ for Ψ, $(x, y) \in \mathrm{K}R^{\mathfrak{I}}$.

3 An Interface-Based Framework for Modular Ontologies

3.1 Formalization

In this section, we introduce the interface-based formalization of modular ontologies and highlight its main features using the following example.

Example 1 *Consider a case where we want to develop an ontology for the tourism domain. We have found an ontology describing different places in Canada and an ontology which covers North America. We desire to utilize these existing ontologies rather than gathering and categorizing geographical information regarding these places in the tourism ontology from scratch. In addition, we want to have a reliable way to understand and use the main features of the these ontologies without being required to go through their knowledge bases and figure out their taxonomies and axioms.*

Furthermore, suppose that we introduce the Sightseeing *concept as a notion to represent the places where tourists are interested to visit. Each sight needs to have a name and a specific address, so that a tourist can easily locate it. However, it is possible to specialize the definition of a sight from different perspectives. For example describing from a scientific perspective, a sight refers to a place that exhibits scientific value. This place may relate to one or more branches of* *science and can be visited by various scientists. On the other hand, from the natural perspective, a sight relates to a place with natural attractions, such as beaches, mountains, parks and jungles. Visiting the places in this category is most suitable in certain seasons of the year and the visitors may need to take specialized equipments with them to be able to enjoy their visit. In the tourism ontology, we desire to only know the common sense of sightseeing concept, while its different specializations can be later bound to it based on different requirements.*

Through the interface-based modular ontology formalism, a *'modular ontology'* is defined as a set of *'ontology modules'*(modules) and *'interfaces'*. An interface is a set of knowledge base expressions, which are used in a module but their exact meaning are provided by other ontology modules. An ontology module may utilize or realize a set of interfaces. Referring to Example 1 we can define a modular ontology as a set of ontology modules: Tourism and Canada Destination, where the Canada Destination ontology module 'realizes' the interface concept `Place` and provides its meaning, properties and instances. The interface concept `Place` would be 'utilized' by the Tourism ontology. Definitions 2 and 3 give the formal specifications of interfaces and modules in the proposed formalism.

Definition 2 *An interface I is defined as $I = \langle C_N, R_N, \mathcal{T} \rangle$ where $\mathcal{T}$ is the TBox of the interface and C_N and R_N are sets of concept and role names used in $\mathcal{T}$. I has no ABox assertions. We say an interface I' extends I if $C_N^{I'} \sqsupseteq C_N^I$ and $R_N^{I'} \sqsupseteq R_N^I$ or $\mathcal{T}^{I'} \equiv \mathcal{T}^I \sqcup \alpha$ where α is a set of general inclusion axioms defined using the signature of I'.*

It is easy to see that if I' extends I and I'' extends I', I'' also extends I. We use $Exd(I)$ to denote a set of all interfaces that extends I.

Definition 3 *An ontology module M is defined as $M = \langle \Psi, I_r, I_u \rangle$ where Ψ is the knowledge base and I_r is a set of all interfaces which is realized by M and I_u is a set of all interfaces which is utilized by M. M can be in any description logic language, but it must support nominals.*

We define a module M as consistent with regards to its interfaces iff $\Psi \cup (\bigcup_{i \in I_r} \mathcal{T}_i) \cup (\bigcup_{j \in I_u} \mathcal{T}_j)$ is consistent. A module which utilizes or realizes an interface must be consistent regarding to it. Let P be a concept or role name in an interface I, it is referred in the knowledge base of modules as $I : P$. A module M realizes an interface I, either if $I \in I_r$ or there is an $i \in Exd(I)$ such that $i \in I_r$.

Given an interface, we refer to the module which uses it as utilizer module and the module which gives semantics to its terms as realizer module.

A module which utilizes an interface needs to access the instances provided by the realizer modules. In the interface-based formalism, we follow a query-based approach to augment the semantic of a utilizer module with the individuals provided by the realizer modules. For instance, regarding Example 1, the Tourism ontology module may pose a query to the Canada Destination ontology on the location of an accommodation. Through the proposed formalism, the Tourism KB is augmented with the individuals that are provided by Canada Destination for the Place concept. This augmentation approach brings considerable advantages for the framework. First, after augmenting a module with appropriate individuals from other modules, reasoning engines are not required to take into account the other modules' knowledge bases anymore. Analogous to the idea of knowledge compilation which is employed in [19] for DDL, knowledge augmentation leads to local reasoning instead of reasoning on external modules, which entails lower time complexity. Secondly, The augmentation process does not pose any limitations on the semantics of the module which realizes an interface. This module's semantics can be changed independently of the semantics of those modules which utilize its interfaces. In other words, those modules which require an interface are dependent on the interface of the realizer modules but not vice versa.

In order to augment the domain of a utilizer module, the proposed formalism uses epistemic queries to retrieve the individuals of interfaces' concepts and roles from the realizer module. The hypothesis behind this approach is that a utilizer module looks at the realizers as black-boxes whose knowledge about the interface terms are compete enough for reasoning. As an explanatory example assume that in the case of Example 1, the Tourism ontology uses an interface concept `SingleLingualCity` which refers to those cities that have only one official language:

$$\texttt{SingleLingualCity} \equiv \texttt{City} \sqcap \leq 1\ \texttt{hasOfficialLang.Language}$$

North America Destination ontology realizes this interface concept with the following expressions in its knowledge base:

```
City(New York),
hasOfficialLang(New York, English),
Language(English)
```

From the point of view of the Tourism ontology, the knowledge base of North America is complete for reasoning about places and cities, so New York will be recognized as a `SingleLingualCity`. Observe that without using epistemic queries, `SingleLingualCity` would not match any instance from the realizer module.

Based on the definition of interfaces and modules, we now define a modular ontology as follows:

Definition 4 *A modular ontology is a triple* $O = \langle \mathrm{M}, \mathrm{I}, F \rangle$ *where* M *is a set of ontology modules,* I *is a set of interfaces whose description logic is less or equally expressive with regards to the description logic of the ontology modules (description constructors of any given interface is the subset of the description constructors of any ontology module) and* F *is a configuration function* $F : \mathrm{M} \times \mathrm{I} \rightarrow \mathrm{M}$ *which chooses one realizing module for every utilizer module-Interface pair.* $F(M, I) = M'$ *if:*

(c1) $I \in I_u^M$ *and* ($I \in I_r^{M'}$ *or there is an* $i \in Exd(I)$ *such that* $i \in I_r^{M'}$)
(c2) M *and* M' *are consistent regarding* I *and* i
(c3) *Let* C_i *and* R_j *be the result sets of queries* $\mathrm{K}\,I : C_i$ *and* $\mathrm{K}\,I : R_j$ *posed to* M', $\Psi_M \bigcup_{C_i \in I}(I : C_i \equiv C_i) \bigcup_{R_j \in I}(I : R_j \equiv R_j)$ *is consistent.*

A path $PATH$ *in an modular ontology is defined as a set of modules which are connected through the configuration function* F*.* $PATH(M)$ *specifies the path which includes module* M*.*

Based on Definition 4, the final form of a modular ontology is specified by the configuration function F. This function shows the connected modules thorough interfaces and its value can be set at configuration time. Introduction of the configuration function F in Definition 4 implies the development and configuration times of a modular ontology are distinguishable. The development time is when an ontology module is developed, its necessary interfaces as well as those interfaces that it realizes are specialized. The configuration time is the time when the required modules are selected to realize the interfaces of a particular module. (e.g. someone may develop the Tourism ontology through the proposed framework and specify that it needs the place interface concept. However, at configuration time it would be finalized whether the Canada ontology or the North America ontology will realize the place concept).

For being connected through the configuration function, two ontology modules should satisfy the three conditions mentioned in Definition 4. First of all a module should realize an interface or one of its extension in order to be selected by the configuration function and be connected to the utilizer module. Secondly, two modules should be consistent with regards to their interfaces. And finally, the third condition ensures that the integration of two modules does not entail inconsistencies. Since the domain of the utilizer module would be augmented by the individuals of the interface terms from the realizer modules through epistemic K queries, condition three ensures that this augmentation does not lead to an inconsistency in the utilizer module. Example 5 shows the formal representation of the situation which is described in Example 1.

Example 5 *Regarding Example 1, Let 'Tourism' be an ontology module which utilizes the interface 'Location'. Furthermore let the signature and TBox of them be as follows:*

$C_N^{Tourism}$=\{`Accommodation`\}
$R_N^{Tourism}$=\{`hasAddress`\}
$C_N^{Location}$=\{`Place`\}

The ontology module Tourism can use the interface terms for creating complex concepts and for defining general inclusion axiom. For instance the TBox of the tourism ontology has the following axiom:

`Accommodation`$\sqsubseteq$ $\exists$`hasAddress.Location:Place`.

Let 'Canada Destination' (CD) and 'North America Destination' (NAD) be two ontology modules which realize the interface Location. Two different values for the configuration F lead to two different modular ontologies O1 and O2 as follows:

O1={ { `Tourism, CD, NDA`},{`Location`}, $F1$ },
where F1(`Tourism`,`Location`*)=CD*
O2={ { `Tourism, CD, NDA`},{`Location`}, $F2$ },
where F2(`Tourism`,`Location`*)=NDA*

3.2 Augmented Semantics

To give a formal specification for the notion of augmentation for an ontology module, we define an augmentation function as follows:

Definition 6 *Let $PTBox(M)$ be a set of all axioms in the TBox of all interfaces in $PATH(M)$ for a given ontology module M, an augmentation function $Aug : M \rightarrow M$ is a function such that $\Psi^{Aug(M)}$ is defined as the union of the following elements:*

(i) $\mathcal{T}^M$,
(ii) $PTBox(M)$,
(iii) $\{I : c_1\} \sqcup \ldots \sqcup \{I : c_n\}$ *where c_i is a member of the result set of* $\mathrm{K}I : C$ *posed to $F(M, I)$ for all concepts C in all $I \in I_u^M$,*
(iv) $\{I : x_1\} \sqcup \ldots \sqcup \{I : x_n\} \sqcup \{I : y_1\} \sqcup \ldots \sqcup \{I : y_m\}$ *where $\langle x_i, y_j \rangle$ is a member of the result set of* $\mathrm{K}I : R$ *posed to $F(M, I)$ for all roles R in all $I \in I_u^M$.*

Based on Definition 6, the result set of the epistemic queries posed to the realizer module is being inserted to the knowledge base of the utilizer module as new nominals. The following definition gives the exact semantic of the augmented module.

Definition 7 *An augmented semantics for a module M_j in a modular ontology $O = \langle \mathrm{M}, \mathrm{I}, F \rangle$, is defined as $\mathcal{I}_j = (\triangle^{\mathcal{I}_j}, \cdot^{\mathcal{I}_j})$, where $\triangle^{\mathcal{I}_j}$ is a non-empty domain for $Aug(M_j)$ and a mapping function $\cdot^{\mathcal{I}_j}$ which maps each concept of $Aug(M_j)$ to a subset of $\triangle^{\mathcal{I}_j}$, each role of $Aug(M_j)$ to a subset $\triangle^{\mathcal{I}_j} \times \triangle^{\mathcal{I}_j}$ and each individual name from $Aug(M_j)$ to an element $a^{\mathcal{I}} \in \triangle^{\mathcal{I}_j}$. $\cdot^{\mathcal{I}_j}$ maps M_j concept expressions based on the semantic of concept constructors of M_j. For the concepts and roles of the utilized interfaces, the function develops mapping as follows:*

(i) For every interface concept $I : C$, $x \in (I : C)^{\mathcal{I}_j}$ iff $\{x\}^{\mathcal{I}_j} \subseteq \triangle^{\mathcal{I}_j}$ and $\Psi_k \vDash \mathrm{K}C(x)$, where $\mathrm{K}C$ is an epistemic query posed to $M_k = F(M_j, I)$,
(ii) For every interface role $I : R$, $\langle x, y \rangle \in (I : R)^{\mathcal{I}_j}$ iff $\{x\}^{\mathcal{I}_j} \subseteq \triangle^{\mathcal{I}_j}$ and $\{y\}^{\mathcal{I}_j} \subseteq \triangle^{\mathcal{I}_j}$ and $\Psi_k \vDash \mathrm{K}R(\langle x, y \rangle)$, where $\mathrm{K}R$ is an epistemic query posed to $M_k = F(M_j, I)$.

An ontology module is augmentedly consistent if there is an augmented interpretation $\mathcal{I}$ (augmented model), which satisfies all axioms and assertions in $\Psi_{Aug(M)}$. Let α be an inclusion axiom or an ABox assertion $\Psi_{Aug(M)} \models \alpha$ if for every augmented model $\mathcal{I}$, α is satisfied by $\mathcal{I}$. For a concept expression α, $\Psi_{Aug(M)} \models \alpha$ if for every augmented model $\mathcal{I}$, α is satisfiable.

In order to augment the knowledge base of a module, we exploit the notion of nominals. The conditions (i) and (ii) of the Definition 7 ensure that the interface concepts and roles in a utilizer module is interpreted the same as their interpretation in the realizer modules. For instance in the situation described in Example 5, suppose the realizer module expresses that {`Toronto`, `Montreal`, `Vancouver`} individuals are of the type of the `Place` concept. These places are inserted as nominal to the domain of the Tourism ontology module and the concept `Location:Place` is interpreted in such way to be equal to {`Toronto`, `Montreal`, `Vancouver`}.

Let us make two remarks about the proposed semantics. First, since the augmented semantic is defined for an ontology module, the $\top$ and $\neg$ symbols in a modular ontology are interpreted from the point of view of each augmented module. For example $\neg I : C$ in a utilizer module M_i is interpreted as $\triangle^{\mathcal{I}_i} \setminus (I : C)^{\mathcal{I}_i}$ when $I : C$ has been interpreted to be equal to the result set of K$I : C$ posed to $F(M_i, I)$. Second, we do not make a *unique name assumption* and hence two nominals may refer to the same individual; therefore, an inserted nominal to a module can be interpreted to be equivalent to an existing nominal in its knowledge base.

4 Properties of the Interface Based Formalism

In this section we describe the significant properties of the proposed modularization framework. Initially we point out two features, 'directed semantic' [1] and 'polymorphism' which are driven from the interface base nature of the formalism. Secondly we prove decidability of the formalism and its capability to propagate the logical consequences of the public parts of inter-connected modules to each others.

4.1 Directed semantic

In the previous section, we pointed out that according to the proposed interface-based modular formalism, the realizer modules are semantically independent from those modules which utilize their interfaces . The importance of such independency is brightened when we observe that it leads to 'directed semantic' [1]. Directed semantic means that if a module which uses a set of interface terms, gives new semantics to these terms, this semantic does not affect their meaning in the original modules. For instance, in the case of Example 1, suppose that the Tourism ontology uses the interface concept `BeautifulPlaces` and `RuralPlaces` and the modular ontology has been configured in such way that the Canada Destination realizes these two interface concepts. The Tourism ontology may add an inclusion axiom that `BeautifulPlaces` $\sqsubseteq$ `RuralPlaces`, however, through our formalism, this subsumption does not necessarily hold in the Canadian Destination ontology.

4.2 Polymorphism

The proposed formalism supports *polymorphism* in the development of modular ontologies in the sense that the meaning of an interface term is subject to specialization based on the configuration of the ontology. When one develops an ontology, she only needs to work with general interfaces, and the specialized meaning of interface concepts would be bound to it in configuration time. For instance, in Example 1, we may define the Sightseeing as an interface concept while Natural Sightseeing and Scientific Sightseeing are modeled as two extensions for this concept. The Tourism ontology may configured to use the one of specialized meaning of Sightseeing. Furthermore, it will be possible to extend the meaning of Sightseeing in the future through the introduction of more perspectives without any changes on the syntax of the Tourism ontology. Example 8 shows this capability of the formalism:

Example 8 *Regarding Example 1, suppose that the ontology module 'Tourism' utilizes the interface Attractions which represents the notion of sightseeing and its related properties and concepts. Knowledge base of the Tourism module includes the following axioms:*

$\mathtt{PlaceToGo} \sqsubseteq \mathtt{Attractions{:}Sightseeing}$
Let $ScientificAttractions$ be an extension for Attractions with the following axiom:
$\mathtt{Sightseeing} \sqsubseteq (\forall\ \mathtt{HasScienceBranch.ScienceBranch})$
$\sqcap\ (\exists\ \mathtt{IsVisitedBy.Scientist})$
Further, let $NaturalAttractions$ be another extension for Attractions with the following axioms:
$\mathtt{Sightseeing} \sqsubseteq (\exists \mathtt{HaveBestSeason.Season})$
$\sqcap\ (\exists\ \mathtt{HavePreCondition.Equipment})$,
$\mathtt{Sightseeing} \sqsubseteq \mathtt{Beach} \sqcup \mathtt{Jungle} \sqcup \mathtt{Park} \sqcup \mathtt{mounts}$

Here, $ScientificAttractions$ and $NaturalAttractions$ provide two morphs *(forms) for the* `Sightseeing` *concept, hence the tourism ontology can answer different type of queries related to the concept* `Sightseeing` *based on the type of configuration.*

4.3 Decidability

Consistency checking is a basic problem in description logic knowledge bases to which the other reasoning problems can be reduced. This issue is more vital in the proposed interface-based formalism because different modules can be defined using different description logics; hence their integration may lead to new inconsistency problems. In the following, we show that the consistency checking of augmented modules in a modular ontology is decidable in our proposed formalism.

Lemma 9 [1]

Let M be an ontology module in a modular ontology, such that it supports nominals and there exists an algorithm for creating completion graphs for deciding the consistency of its description logic[2]. *The problem of consistency checking for $Aug(M)$ is decidable.*

Hence, while the formalism provides an expressiveness power which allows creating complex concepts using foreign terms, it ensures that the consistency checking of the integration of different ontology modules is decidable.

4.4 Transitive Logical Consequences and Partial Reuse

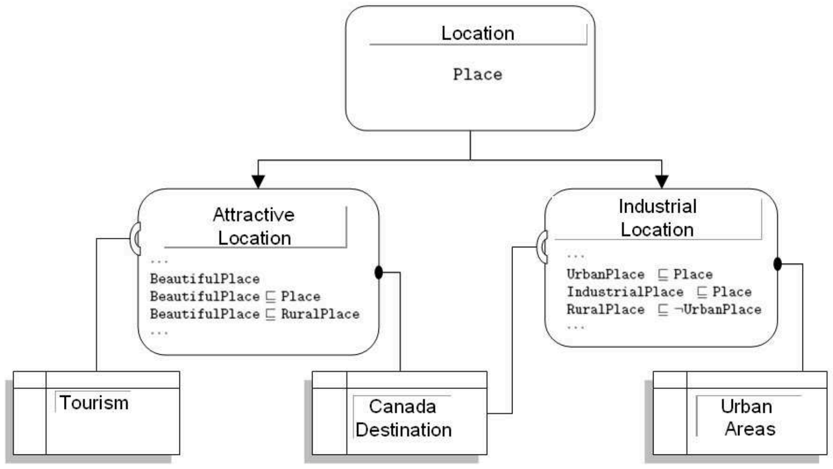

Fig. 1. An example of propagating logical consequences in the interface-based modularization formalism

The interface-based formalism supports propagating the logical consequences of the public part of an ontology module to all of the connected ontology modules, while its private parts do not have such consequence propagation. Hence, the formalism does not require the ontology modules to use or import all of the terms of the other connected modules and does not require reasoning engines to process the union of all inter-connected modules. The public section of each ontology module is the knowledge base of its interfaces. Lemma 10 shows that the logical consequences of a module's interfaces propagate to all other modules that are connected to it.

[1] For the proofs of all discussed lemmas in this paper see: http://falcon.unb.ca/ ∼ m4742/mo.pdf

[2] Such algorithms can be found in [12, 11].

Lemma 10

(1) Let $M_1 = \langle \mathcal{T}, I_u, I_r \rangle$ be an augmentedly consistent module in a modular ontology, $I_1 = \bigcup_{I \in (I_u, I_r)} I$ and α be a concept expression (General inclusion axiom) whose signature is a subset of the signature I_1. If $I_1 \models \alpha$, for all augmentedly consistent module such as M on $PATH(M_1)$, $\Psi_{Aug(M)} \models \alpha$.

(2) Furthermore, consider an augmentedly consistent module M_2 on $PATH(M_1)$ and let I_2 be defined for M_2 similarly to I_1 for M_1. Let $I : A$ and $I : B$ be two concepts in I_1 and $I : C$ a concept in I_2. If $I_1 \models I : A \sqsubseteq I : B$ and $I_2 \models I : B \sqsubseteq I : C$, for all augmentedly consistent modules such as M on $PATH(M_1)$, we have $\Psi_{Aug}(M) \models I : A \sqsubseteq I : C$.

As an explainer example, consider the situation shown in Figure 1. According to the figure, there are three ontology modules: Tourism, Canada Destination and Urban Areas and three interfaces: Location, Attractive Location and Industrial Location where the last two interfaces are extensions of the Location interface. Tourism and Canada Destination utilize Attractive Location and Industrial Location, respectively. Moreover Canada Destination realizes Attractive Location and Urban Areas realizes Industrial Location. Since Industrial Location is the public part of the Urban Areas ontology modules, all of its axioms is propagated to the connected ontology modules. According to Industrial Location `RuralPlace` $\sqsubseteq \neg$`UrbanPlace`, and since Canada destination has `BeautifulPlace` $\sqsubseteq \neg$`RuralPlace` in its own interface, its augmented semantics entails `BeautifulPlace` $\sqsubseteq \neg$`UrbanPlace`.

The following proposition shows the unsatisfiability of a concept in a realizer module will be preserved in all of its utilizer modules.

Proposition 11 *Let M be a module which realizes interface I, for all modules M' such that $F(M', I) = M$, $\Psi_{Aug(M')} \models I : C \sqsubseteq \perp$, if $\Psi_{Aug(M)} \models I : C \sqsubseteq \perp$*

Proof Sketch: For every unsatisfiable concept $I : C$, the result set of the query K$I : C$ is the empty set.

In contrast to the public section, the private section of an ontology module does not necessarily propagate monotonically through connected modules. For example in the case of Figure 1, consider that Urban Areas module indicates that `IndustrialPlace(Toronto)`,`UrbanPlace(Toronto)` and `UrbanPlace(Montreal)` in its ABox. The Canada destination ontology module may conclude that `IndustrialPlace` is a subclass of `UrbanPlace` even though this axiom does not necessary hold in the realizer module Urban Areas.

5 Implementation

In this section, we present the implementation of the interface-based formalism introduced in the previous sections. The objective of this implementation is to allow ontology developers to define a modular ontology based on the definitions provided by the formalism as a set of ontology modules and interfaces, configure

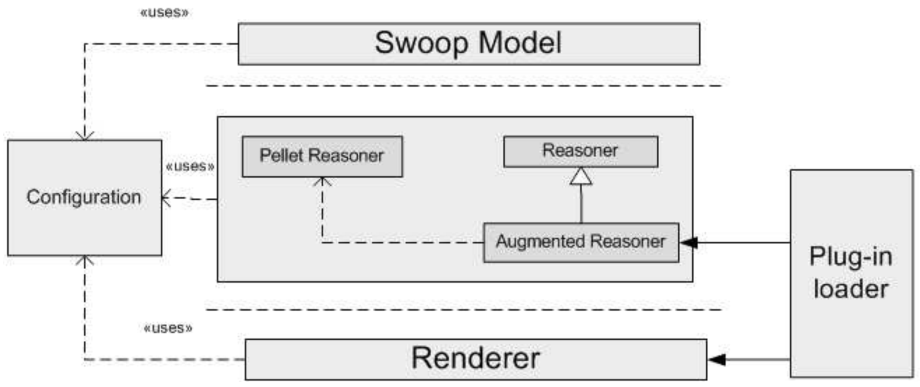

Fig. 2. A new extension to the architecture of Swoop for supporting interface-based modular ontologies

the modular ontology and select the connected ontologies, and also be able to perform reasoning on the developed modular ontology.

For the purpose of the implementation, we perform two tasks. First, we extend OWL-DL in order to allow ontology modules to use or realize a set of interfaces and second we extend the architecture of the Swoop ontology editor and browser in order to be able to work with interfaces and perform reasoning on the modular ontologies based on the semantics described earlier.

In order to extend OWL-DL, we define 'useInterface' and 'realizeInterface' as two new built-in ontology properties, analogous to the definition of 'owl:imports'. 'useInterface' and 'realizeInterface' are followed by the ID of interfaces they use and realize, respectively. We also modify OWL-API, a set of java interfaces for manipulating owl ontologies, such that an 'ontology' object has references to its used and realized interfaces.

Swoop [13] is an ontology browser and editor which is tailored for OWL ontologies. It provides a convenient environment for manipulating multiple ontologies. The architecture of Swoop is comprised of three components: Model, Reasoning and Rendering. In addition, it consists of a plug-in loader which loads the appropriate reasoner or renderer in the environment. We modify the architecture of Swoop by introducing an augmented reasoner as well as a configuration object which can be shared among different layers of the architecture. Figure 2 depicts the modified architecture of the Swoop ontology editor.

As it is illustrated in Figure 2, the augmented reasoner can be defined as an extension to any existing reasoner available for Swoop. The augmented reasoner augments the knowledge base of the ontology module with the result set of the epistemic queries posed to the modules which realize its required interfaces before performing a reasoning task. The augmented reasoner uses the capability of performing epistemic queries from Pellet for doing its augmentation process. It uses the configuration component in order to recognize the appropriate realizer modules.

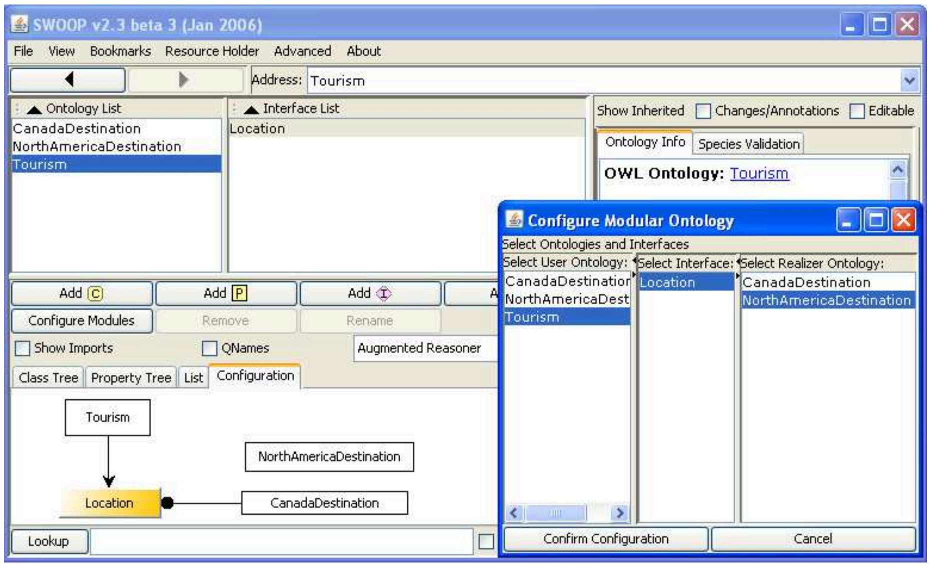

Fig. 3. Interface-based modular ontologies in Swoop

We also modify the Model component of the Swoop architecture such that it provides capabilities for loading and working with interfaces as well. Using the extended OWL-API, the new version of Swoop supports loading interfaces and configuring modular ontologies in such a way that for each ontology module the users can select a realizer module for each of the interfaces it uses. Figure 3 shows a snapshot of the modified Swoop environment for creating a modular ontology.

As it can be seen it this figure, a user can graphically see the configuration of the modular ontology through the newly introduced tab: ”Configuration”. Moreover, clicking on the ”Configure Module” button, a pop-up menu is shown to the user that contains the list of all modules and for each selected module, the list of all interfaces it uses and for any selected interface the list of its realizer modules. The users can use this menu to change the configuration of a modular ontology.

6 Related Work

We can categorize the accomplished efforts on modular ontologies into two classes: (*i*) those that attempt to decompose a large and comprehensive ontology into a set of smaller and self-contained modules and (*ii*) those that introduce new formalisms for developing modular ontologies.

With regards to the first category, [6] proposes an algorithm for extracting a module from an ontology which describes a given concept. The extracted module captures the meaning of that concept and should be ‘locally correct’ and ‘locally complete' which means that any assertions which are provable in a module should also be provable in the ontology. Also any assertion which is provable in the ontology and asserted using the signature of a module should be provable in that module. The notion of local complete modules is close to the notion of 'conservative extensions' which is discussed in [8] and employed for extracting an approximation of the smallest meaningfull module related to a set of concept and role names from an ontology. [16] proposes an algorithm for segmentation of an ontology by traversing through the ontology graph starting from a given concept name. In [20], authors propose a method for partitioning a large ontology into disjoint sets of concepts. It has been assumed that the given ontology is mostly comprised of hierarchal relationships between concepts instead of more complex roles and binary relationships.

The efforts in the second category focus on proposing new formalisms for modular ontologies. These formalisms mostly provide new extensions to existing description logics syntax and semantics in order to make automated reasoning over ontology modules feasible. The interface-based modularity formalism introduced in this paper and also E-connections, DDL, P-DL and semantic import which are discussed in the first section of this paper can be categorized in this class of works. [15] describes a new formalism for ontology modularization. Based on [15] a module is described by its identifier, a set of the identifiers of other modules which are imported, a set of interfaces, a set of mapping assertions between different concept names of imported modules and its concepts and an export interface. It uses a mapping approach and defines a mapping function in order to let a module use the others' elements. Through this function, concepts of different modules are mapped to a global domain. The notion of interfaces in [15] are different from those that are defined in this paper. Contrary to our approach for defining interfaces as independent ontology units which ensures indirect connection of ontology modules, in [15] interfaces are a set of concept names of a specific module that can be imported by others. Furthermore, in [15] a module can only have one export interface, consequently it is not possible to provide different perspectives for describing an ontology module.

7 Concluding Remarks

In this paper, we have described a framework for developing ontologies in a modular manner. The core of this framework is the interface-based modular ontology formalism which supports knowledge encapsulation, i.e., it allows ontology modules to describe their content through well-defined interfaces. We showed that this formalism provides a considerable expressiveness power by allowing ontology modules to create complex roles and concepts using the interface terms. We have also proven the decidability of the formalism and its capability for propagating the logical consequences of the public parts of the ontology modules. In order to make the application of the frameworks feasible in real-world applications, we have extended the syntax and semantic of OWL-DL and the Swoop architecture.

References

1. Bao, J., Caragea, D., and Honavar, V. On the semantics of linking and importing in modular ontologies. In *International Semantic Web Conference* (2006), pp. 72–86.
2. Bao, J., Slutzki, G., and Honavar, V. A semantic importing approach to knowledge reuse from multiple ontologies. In *AAAI* (2007), pp. 1304–1309.
3. Borgida, A., and Serafini, L. Distributed description logics: Assimilating information from peer sources. *J. Data Semantics 1* (2003), 153–184.
4. Bouquet, P., Giunchiglia, F., Harmelen, F., Serafini, L., and Stuckenschmidt, H. C-OWL: Contextualizing ontologies. In *ISWC 2003* (2003).
5. Calvanese, D., Giacomo, G. D., Lembo, D., Lenzerini, M., and Rosati, R. Eql-lite: Effective first-order query processing in description logics. In *IJCAI* (2007), pp. 274–279.
6. Cuenca Grau, B., Parsia, B., Sirin, E., and Kalyanpur, A. Modularity and web ontologies. In *Proceedings of KR-2006* (2006), AAAI Press, pp. 198–209.
7. Donini, F. M., Lenzerini, M., Nardi, D., Nutt, W., and Schaerf, A. An epistemic operator for description logics. *Artif. Intell. 100*, 1-2 (1998), 225–274.
8. Grau, B. C., Horrocks, I., Kazakov, Y., and Sattler, U. Just the right amount: extracting modules from ontologies. In *WWW '07* (2007), ACM, pp. 717–726.
9. Grau, B. C., and Kutz, O. Modular ontology languages revisited. In *Proceedings of the IJCAI'07 Workshop on Semantic Web for Collaborative Knowledge Acquisition,* (2007).
10. Grau, B. C., Parsia, B., and Sirin, E. Combining owl ontologies using e-connections. *Journal of Web Semantics 4*, 1 (2005).
11. Horrocks, I., and Sattler, U. Ontology reasoning in the shoq(d) description logic. In *IJCAI* (2001), pp. 199–204.
12. Horrocks, I., Sattler, U., and Tobies, S. Practical reasoning for expressive description logics. In *LPAR '99* (London, UK, 1999), Springer-Verlag, pp. 161–180.
13. Kalyanpur, A., Parsia, B., Sirin, E., Grau, B. C., and Hendler, J. A. Swoop: A web ontology editing browser. *J. Web Sem. 4*, 2 (2006), 144–153.
14. Kutz, O., Lutz, C., Wolter, F., and Zakharyaschev, M. E-connections of abstract description systems. *Artif. Intell. 156*, 1 (2004), 1–73.
15. Peter Haase, Sebastian Rudolph, J. E. A. Z. M. D. M. I. Y. J. C. C. C. B. A. J. M. G. Deliverable d1.1.3 neon formalisms for modularization: Syntax, semantics, algebra, 2008. NEON EU-IST-2005-027595.
16. Seidenberg, J., and Rector, A. Web ontology segmentation: analysis, classification and use. In *WWW '06* (New York, NY, USA, 2006), ACM, pp. 13–22.
17. Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., and Katz, Y. Pellet: A practical owl-dl reasoner. *Web Semant. 5*, 2 (2007), 51–53.
18. Snyder, A. Encapsulation and inheritance in object-oriented programming languages. In *OOPLSA '86* (New York, NY, USA, 1986), ACM, pp. 38–45.
19. Stuckenschmidt, H., and Klein, M. C. A. Integrity and change in modular ontologies. In *IJCAI* (2003), pp. 900–908.
20. Stuckenschmidt, H., and Klein, M. C. A. Structure-based partitioning of large concept hierarchies. In *International Semantic Web Conference* (2004), pp. 289–303.