Combining OWL Ontologies Using $\mathcal{E}$-Connections

Bernardo Cuenca Grau [a,b] , Bijan Parsia [a] , Evren Sirin [a] ,

[a] *University of Maryland, MIND Lab, 8400 Baltimore Ave, College Park MD 20742, USA*

[b] *Departamento de Informatica, Universidad de Valencia Av. Vicente Andres Estelles, s/n, 46100 Burjassot, Valencia, SPAIN*

Abstract

The standardization of the Web Ontology Language, OWL, leaves (at least) two crucial issues for Web-based ontologies unsatisfactorily resolved, namely how to represent and reason with multiple distinct, but linked ontologies, and how to enable effective knowledge reuse and sharing on the Semantic Web.

In this paper, we present a solution for these fundamental problems based on $\mathcal{E}$-Connections. We aim to use $\mathcal{E}$-Connections to provide modelers with suitable means for developing Web ontologies in a modular way and to provide an alternative to the *owl:imports* construct.

With such motivation, we present in this paper a syntactic and semantic extension of the Web Ontology language that covers $\mathcal{E}$-Connections of OWL-DL ontologies. We show how to use such an extension as an alternative to the *owl:imports* construct in many modeling situations. We investigate different combinations of the logics $\mathcal{SHIN}(\mathcal{D}), \mathcal{SHON}(\mathcal{D})$ and $\mathcal{SHIO}(\mathcal{D})$ for which it is possible to design and implement reasoning algorithms, well-suited for optimization.

Finally, we provide support for $\mathcal{E}$-Connections in both an ontology editor, SWOOP, and an OWL reasoner, Pellet.

Key words: Web Ontology Language, Integration and Combination of Ontologies, Combination of Knowledge Representation Formalisms, Description Logics reasoning

Email addresses: bernardo@mindlab.umd.edu (Bernardo Cuenca Grau), bparsia@isr.umd.edu (Bijan Parsia), evren@cs.umd.edu (Evren Sirin).

1 Motivation

The Semantic Web architecture has been envisioned as a set of new languages that are being standardized by the World Wide Web Consortium (W3C). Among these languages, the Web Ontology Language (OWL) plays a prominent role, and Description Logics have deeply influenced its design and standardization (1)(2). Two of the three variants, or dialects, of OWL, namely OWL-Lite and OWL-DL, correspond to the logics $\mathcal{SHIF}(\mathcal{D})$ and $\mathcal{SHOIN}(\mathcal{D})$, respectively (2) (3) (4).

The acceptance of OWL as a Web standard will yield to the rapid proliferation of DL ontologies on the Web and it is envisioned that, in the near future, the Semantic Web will contain a large number of independently developed ontologies.

However, the standardization of OWL also leaves (at least) two crucial issues for Web-based ontologies unsatisfactorily resolved, namely how to represent and reason with multiple distinct, but linked ontologies, and how to enable effective knowledge reuse and sharing on the Semantic Web.

First, in order to provide support for integrating Web ontologies, OWL defines the *owl:imports* construct, which allows to include by reference in a knowledge base the axioms contained in another ontology, published somewhere on the Web and identified by a global name (a URI). However, the functionality provided by this construct is unsatisfactory for a number of reasons (5):

- The only way that the *owl:imports* construct provides for using concepts from a different ontology is to bring into the original ontology *all* the axioms of the imported one. Therefore, the only difference between *copying and pasting* the imported ontology into the importing one and using an *owl:imports* statement is the fact that with imports both ontologies stay in different files. This certainly provides some *syntactic* modularity, but not a *logical* modularity, which would be indeed more desirable.
- The components of an ontology, such as classes and properties, are, as the ontologies themselves, identified by unique names (URIs) on the Semantic Web. For example, suppose that we are developing an ontology about "People" and we want to define the concept of a "Dog Owner". It may seem natural for such a purpose to use the URI of a certain class "Dog", that appears in an ontology about "Pets" that we have found on the Web. We may think then that we are committing to the meaning of "Dog" in that ontology, i.e., that a dog is an animal, for example. However, if we use the URI for "Dog" without importing the corresponding ontology, we are bringing *nothing* from the meaning of the term in the the foreign ontology, while if we import it, we are bringing all the axioms of the "Pet" ontology

to our logical space, even if we are only interested in dogs, and not in cats or hamsters.
- The use of *owl:imports* results in a completely flat ontology, i.e., none of the imported axioms or facts retain their context. While it is possible to track down the originator(s) of some assertions by inspecting the imported ontology, OWL reasoning does not take such context into account.

Hence, in OWL, we can let in either all the axioms of a foreign ontology, or none.

Second, enabling knowledge reuse and sharing has been, since the very birth of OWL, a major goal of the Web Ontology Working Group. Ontology engineering is a highly time-consuming task. As more ontologies are built and become available, and as the size of ontologies grows, knowledge sharing and reuse become crucial research issues.

On the one hand, when ontologies grow, they become harder for the reasoners to process and for humans to understand, and also harder to reuse. On the other hand, as more ontologies become available, the advantages of reusing existing ontologies become more apparent. In order to make reuse and sharing easier, ontologies should be designed as mostly independent and self-contained modules (6)(7). Intuitively, a module should contain information about a self-contained subtopic, i.e., an application domain that can largely stand for itself. Then, suitable means should be provided for integrating and connecting those modular ontologies.

In this paper, we present an approach for tackling these fundamental problems based on $\mathcal{E}$-Connections. The $\mathcal{E}$-Connections technique (8) (9) is a method for combining logical languages that are expressible in the Abstract Description System (ADS) framework. ADSs (10) are a generalization of description, modal and epistemic logics and many logics of time and space. The main motivation of $\mathcal{E}$-Connections is to combine decidable logics in such a way that the resulting combined formalism remains decidable, although the increase of expressivity may result in a higher worst-case complexity.

With such a motivation, we present in this paper a syntactic and semantic extension of the Web Ontology language that covers $\mathcal{E}$-Connections of OWL-DL ontologies. We show how such an extension can be used to achieve modular ontology development on the Semantic Web and how $\mathcal{E}$-Connections provide a suitable framework for integration of Web ontologies. We investigate the use of $\mathcal{E}$-Connections as an alternative to the *owl:imports* construct in many modeling situations.

We explore different combinations of the logics $\mathcal{SHIN}(\mathcal{D}), \mathcal{SHON}(\mathcal{D})$ and $\mathcal{SHIO}(\mathcal{D})$, which stand at the basis of OWL. We show that, for these combinations, it is possible to design reasoning algorithms, well-suited for implemen-

tation and optimization. We prove that these algorithms can be implemented as an extension of current reasoners and, contrary to what is thought about $\mathcal{E}$-Connections, we argue that they have a potential for enhancing performance, since they suggest new optimization techniques.

Finally, we provide support for $\mathcal{E}$-Connections in both an OWL ontology editor and an OWL reasoner. Our aim has been to build an $\mathcal{E}$-Connection aware infrastructure, by extending our ontology editor SWOOP (11) and our reasoner Pellet (12), that people with a large commitment to OWL will find understandable and useful. We have been mostly concerned with reusing as much of our current tool support for OWL as possible. Our experience in implementing OWL tools has taught us that implementing a Semantic Web Knowledge Representation formalism is different from implementing the very same formalism outside the Semantic Web framework. Thus, we aim to explore both the issues related to the implementation of a new KR formalism, $\mathcal{E}$-Connections, and those concerning its integration in a Semantic Web context.

2 $\mathcal{E}$-Connections of Web Ontologies

An $\mathcal{E}$-Connection is a knowledge representation language defined as a combination of other logical formalisms. Each of the component logics has to be expressible in the Abstract Description System (ADS) framework (10), which includes Description Logics (and hence OWL-DL), some temporal and spatial logics, Modal and Epistemic logics. Obviously, different component logics will give rise to different combined languages, with different expressivity and computational properties.

$\mathcal{E}$-Connections were originally introduced in (8) as a way to go beyond the expressivity of each of the component logics, while preserving the decidability of the reasoning services. Thus, $\mathcal{E}$-Connections were conceived for providing a trade-off between the expressivity gained and the computational robustness of the combination.

Here, we will use $\mathcal{E}$-Connections as a language for defining and instantiating combinations of OWL-DL ontologies. We will restrict ourselves to OWL-DL, since OWL-Full is beyond the Abstract Description System framework. From now on in the paper, whenever we mention *OWL*, we will implicitly refer to OWL-DL.

An $\mathcal{E}$-Connection is a set of "connected" ontologies. An $\mathcal{E}$-Connected ontology [1] typically contains information about classes, properties and their in-

[1] In this paper, we use "$\mathcal{E}$-Connection Language" to denote a formalism, i.e. a logic;

stances, as in OWL, but also about a new kind of properties, called *link properties*, which are somewhat similar in spirit to datatype properties.

In OWL, the classes defined in terms of datatype properties "combine" information from different domains: the actual application domain of the ontology and the domain of datatypes. The coupling between datatypes and the ontology is always achieved through restrictions on datatype properties. For example, a "retired person" can be defined in OWL as a person whose age is greater than 65, by using a class ("Person") in the ontology and a restriction on a datatype property "age" with value "greater than 65". Both from a logical and from a modeling perspective, the domain of the ontology and the domain of datatypes are disjoint: from a modeling perspective, the (application) domain of "persons" is disjoint from the (application) domain of "numbers"; from a logical perspective, in OWL, the domain where classes, properties and individuals in the ontology are interpreted is disjoint from the domain of datatypes, and datatype properties are interpreted as binary relations with the first element belonging to the domain of the ontology and the second on the domain of the datatypes.

In the same vein, link properties allow to create classes in a certain ontology based on information from a *different* ontology, provided that the domains of the ontologies are disjoint, both from a logical and a modeling perspective. For example, a *GraduateStudent* in an ontology about "people" could be defined as a student who is enrolled in at least one graduate course, by using the class *Student* in the people ontology and a *someValuesFrom* restriction on the link property *enrolledIn* with value *GraduateCourse*, which would be a class in a different ontology dealing with the domain of "academic courses".

Link properties are logically interpreted as binary relations, where the first element belongs to its "source" ontology and the second to its "target ontology". Conceptually, a link property will be defined and used in its "source" ontology. For example, the link property "enrolledIn" would be defined as a link property in the "people" ontology with target ontology "academic courses".

An $\mathcal{E}$-Connected ontology can be roughly described as an OWL-DL ontology, extended with the ability to define link properties and construct new classes in terms of restrictions on them. An $\mathcal{E}$-Connection is then defined as a set of $\mathcal{E}$-Connected ontologies.

From the modeling perspective, each of the component ontologies in an $\mathcal{E}$-Connection is modeling a different application domain, while the $\mathcal{E}$-Connection itself is the (disjoint) *union* of all these domains. For example, an $\mathcal{E}$-Connection

---

we will use "$\mathcal{E}$-Connection" to denote a knowledge base written in such a language. These knowledge bases are composed of a set of "connected" ontologies, for which we will use the term "$\mathcal{E}$-Connected ontology" or "component ontology".

could be used to model all the relevant information referred to a certain university, and each of its component ontologies could model, respectively, the domain of people involved in the university, the domain of schools and departments, the domain of courses, etc.

2.1 Basic Elements

In order to illustrate the basic elements of an $\mathcal{E}$-Connection, let us consider the following application domains, that we want to formalize: let D_1 be the domain of "travel accommodations", D_2 the domain of "leisure activities", D_3 the domain of "travel destinations", and D_4 the domain of "people". We want to use an $\mathcal{E}$-Connection to model the union of these domains, i.e., the domain of "tourism".

We want to model each application domain in a different component of the $\mathcal{E}$-Connection and then use link properties to talk about their relationships.

As in OWL, each $\mathcal{E}$-Connected ontology is written in a different file, and the vocabularies being used in each of them can be specified by a set of namespace declarations and entity definitions.

Each $\mathcal{E}$-Connected ontology defines its own ontology header, and may include a collection of assertions about the ontology under an *owl:Ontology* tag. As in OWL, these tags may contain meta-data about the component ontologies, such as comments information about version control, and also *owl:import* statements. The *rdf:about* attribute provides a name (a URI) for each ontology.

An $\mathcal{E}$-Connected ontology can import another ontology. Note that, although $\mathcal{E}$-Connections can replace import statements in many situations, sometimes it makes more sense to import than to $\mathcal{E}$-connect [2].

For our domains, we create the following root classes [3]:

```
(accommodations)
<owl:Class rdf:ID= "Accommodation"/>

(activities)
<owl:Class rdf:ID= "Activity"/>
```

[2] We will investigate later on in the paper the interaction between imports and $\mathcal{E}$-Connections.

[3] In brackets, we specify the ontology each class has been defined in; we use this informal notation along this section for clarity and brevity, in order to avoid including the namespace and ontology headers of each ontology in the combination.

```
(destinations)
<owl:Class rdf:ID= "Destination"/>

(people)
<owl:Class rdf:ID= "Person"/>
```

We would like to define classes like *BudgetDestination* (a travel destination which provides a choice of budget accommodations), a *CaribbeanHotel* (a hotel accommodation offered at a Caribbean destination), and a *SportsDestination* (a destination that offers a variety of activities related to sport).

In order to attain this goal, we define a set of *link properties*, i.e. properties that relate elements of the different domains. For example, the links *providesAccommodation* and *offersActivity* relate the domain of "destinations" to the domain of "accommodations" and "activities" respectively.

```
(destinations)
<owl:LinkProperty rdf:ID="providesAccommodation">
  <owl:foreignOntology rdf:resource="&accommodations;"/>
  <rdfs:domain rdf:resource="#Destination"/>
  <rdfs:range>
    <owl:ForeignClass rdf:about="&accommodations;#Accommodation">
      <owl:foreignOntology rdf:resource="&accommodation;"/>
    </owl:ForeignClass>
  </rdfs:range>
</owl:LinkProperty>
<owl:LinkProperty rdf:ID="offersActivity">
  <owl:foreignOntology rdf:resource="&activities;"/>
  <rdfs:domain rdf:resource="#Destination"/>
  <rdfs:range>
    <owl:ForeignClass rdf:about="&activities;#Activity">
      <owl:foreignOntology rdf:resource="&activities;"/>
    </owl:ForeignClass>
  </rdfs:range>
</owl:LinkProperty>
```

A link property is a binary relation between instances of classes, which belong to different $\mathcal{E}$-Connected ontologies. The *source* of a link property is the ontology in which it has been declared; the *target* of the link is the ontology specified in the *owl:foreignOntology* tag in the declaration.

The first element of the relation always belongs to an instance of a class in the source ontology. In the example, both *providesAccommodation* and *offersAc-*

tivity have been defined in the "destinations" ontology. The second element of the relation corresponds to an individual in the target ontology, i.e. the "accommodations" ontology in the case of *providesAccommodation* and the *activities* ontology in the case of *offersActivity*.

The definition of a link property *must* include a *single owl:foreignOntology* tag. As in the case of object properties, link properties can be assigned a domain and a range. For example, the link property *offersActivity* relates instances of the class *Destination* to instances of the class *Activity*. The class specified as a range of a link property *must* be declared as a class in the target ontology. In the source ontology, such a class can be declared as "foreign" using the *owl:ForeignClass* tag.

A URI cannot be used in a given ontology both as "local" (declared as a class in the ontology using the owl:Class tag) and "foreign"; if this happens, a reasoner *must* treat such an ontology as *inconsistent*.

A link property can be defined as functional or inverse functional, with the usual meaning.

However, as opposed to object properties in OWL, a link property cannot be tagged as transitive or symmetric. Note that, within an $\mathcal{E}$-Connection, a link property is defined in a certain "source" ontology and points to a specific "target ontology". In other words, each link property connects (only) two ontologies in a given $\mathcal{E}$-Connection. Tagging a link property as transitive or symmetric would require the ability to define link properties with several source and target ontologies as well as extending the $\mathcal{E}$-Connections framework in a non-trivial way. Exploring such extensions is part of our ongoing work.

Restrictions on link properties can be used to generate new concepts. For example, we can define a "budget destination" as a travel destination that offers at least one kind of budget accommodation:

```
(destinations)
<owl:Class rdf:ID="BudgetDestination">
  <owl:intersectionOf rdf:parseType="Collection">
     <owl:Class rdf:about="#Destination"/>
     <owl:Restriction>
       <owl:onProperty rdf:resource="#providesAccommodation"/>
       <owl:someValuesFrom>
          <owl:ForeignClass rdf:about="&accommodation;BudgetAccommodation">
             <owl:foreignOntology rdf:resource="&accommodation;"/>
         </owl:ForeignClass>
       </owl:someValuesFrom>
     </owl:Restriction>
```

```
  </owl:intersectionOf>
</owl:Class>
```

Similarly, we can define a *CinemaLover* as a person who likes Cinema:

```
(persons)
<owl:Class rdf:ID="CinemaLover">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Person"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#likesActivity"/>
      <owl:someValuesFrom>
        <owl:ForeignClass rdf:about="&activities;CinemaActivity">
          <owl:foreignOntology rdf:resource="&activities;"/>
        </owl:ForeignClass>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subclassOf>
</owl:Class>
```

Using an "allValuesFrom" restriction we can define a *FanaticCinemaLover* as a *CinemaLover* who likes no activity other than cinema:

```
(persons)
<owl:Class rdf:ID="FanaticCinemaLover">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#CinemaLover"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#likesActivity"/>
      <owl:allValuesFrom>
        <owl:ForeignClass rdf:about="&activities;CinemaActivity">
          <owl:foreignOntology rdf:resource="&activities;"/>
        </owl:ForeignClass>
      </owl:allValuesFrom>
    </owl:Restriction>
  </rdfs:subclassOf>
</owl:Class>
```

where in the "activities" ontology we would define the class *CinemaActivity* as a subclass of *Activity*:

```
(activities)
<owl:Class rdf:ID="CinemaActivity">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Activity"/>
  </rdfs:subClassOf>
</owl:Class>
```

Cardinality restrictions on link properties allow to constrain the number of objects linked by the connecting relations. For instance, we can define a *SportsDestination* as a travel destination that offers more than 10 different sports activities:

```
(destinations)
<owl:Class rdf:ID = "SportsDestination">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Destination"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="offersSportActivity"/>
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">10
      </owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

The "hasValue" restriction on link properties allows to specify classes in an $\mathcal{E}$-Connected ontology based on the existence of a particular individual in a different ontology. For example, we can define the class *SurfingDestination* as a travel destination that offers surfing as one of their activities, where *surfing* is an instance of the class *SportsActivity*.

```
(destinations)
<owl:Class rdf:ID = "SurfingDestination">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Destination"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource = "offersSportActivity"/>
```

```
      <owl:hasValue>
         <owl:ForeignIndividual rdf:about="&activities;surfing">
           <owl:foreignOntology rdf:resource="&activities;"/>
         </owl:ForeignIndividual>
      </owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

where in the "activities" ontology *surfing* is defined to be an individual of the class *SportsActivity*.

```
(activities)
<activities:Surfing rdf:ID="surfing"/>
```

A link property can be defined as the inverse of another link property. For example, *link inversion* would allow to define a *WaterSport* activity a *SportsActivity* that is offered at a *BeachDestination*:

```
(activities)
<owl:Class rdf:ID="WaterSport">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Sport"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isOfferedAt"/>
      <owl:someValuesFrom>
        <owl:ForeignClass rdf:about="&destinations;BeachDestination">
          <owl:foreignOntology rdf:resource="&destinations;"/>
        </owl:ForeignClass>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subclassOf>
</owl:Class>

(activities)
<owl:LinkProperty rdf:ID="isOfferedAt">
  <owl:foreignOntology rdf:resource="&destinations;"/>
  <owl:inverseOf rdf:resource="&destinations;offersActivity"/>
</owl:LinkProperty>
```

A link property may be defined as a sub-property of another link property. For example, we can define *offersSportActivity* as a sub-property of *offersActivity*:

```
(destinations)
<owl:LinkProperty rdf:ID="offersSportActivity">
  <owl:foreignOntology rdf:resource="&activities;"/>
  <rdfs:subPropertyOf rdf:about="#offersActivity"/>
</owl:LinkProperty>
```

Obviously, a link property cannot be declared as a sub-property of an object or datatype property, nor a sub-property of a link relation with a different foreign ontology.

2.2 Axioms and Facts in an $\mathcal{E}$-Connected Ontology

An $\mathcal{E}$-Connected ontology is a sequence of axioms and facts: logical sentences that allow to make assertions about the domain. For example, as in OWL, we can use an axiom to assert that *GraduateStudent* is a subclass of *Student* and a fact to state that *john* is *enrolledIn* the *WebTechnologies* course.

In $\mathcal{E}$-Connected ontologies it is also possible to use a fact to instantiate a link property. For example, we can assert that *SaintThomasIsland* is an instance of *CaribbeanDestination* and that it offers the *surfing* activity:

```
(destinations)
<rdf:Description rdf:about="#SaintThomasIsland">
  <rdf:type>
    <owl:Class rdf:about="#CaribbeanDestination"/>
  </rdf:type>
  <offersActivity>
    <owl:ForeignIndividual rdf:about="&activities;#surfing">
      <owl:foreignOntology rdf:resource="&activities;"/>
    </owl:ForeignIndividual>
  </offersActivity>
</rdf:Description>
```

The $\mathcal{E}$-Connections framework imposes some restrictions to axioms and facts. For example, a class cannot be declared in an ontology as a subclass of a class declared in a foreign ontology in the combination. A property (object, datatype or link) cannot be declared as sub-relation of a foreign property; an individual cannot be declared as an instance of a foreign class, and a pair of individuals cannot instantiate a foreign property.

$\mathcal{E}$-Connections also constrain the use of URIs. In OWL-DL, a URI cannot be used, for example, both as a class and a datatype, or as an object property and a datatype property. In an $\mathcal{E}$-Connected ontology, a set of additional restrictions must be imposed, namely a URI cannot be used "locally" in two different component ontologies. These issues will be addressed in detail later on in the paper.

2.3 What is a $\mathcal{E}$-Connection in a Semantic Web Context?

An $\mathcal{E}$-Connection is a set of $\mathcal{E}$-Connected ontologies. However, this definition can be ambiguous, since it may not be apparent at the first sight to *which* set we are referring to.

For example, let us consider again the "tourism" domain. At first sight, one would say that we have a single $\mathcal{E}$-Connection, namely, the one composed by the ontologies: "destinations", "activities", "accommodations", "people". However, this is not strictly correct.

Suppose that the "people" ontology contains an explicit contradiction. Assume we load the "accommodations" ontology in a reasoner and try to check its consistency; what should the reasoner *do*? The "accommodations" ontology contains no link property declarations, i.e., it is an "ordinary" OWL ontology. In this case, an $\mathcal{E}$-Connections aware reasoner should *only* check the consistency of that ontology, and ignore the rest, i.e., it should not report the inconsistency in the "people" ontology.

Given an $\mathcal{E}$-Connected ontology, a reasoner should consider *only* all the $\mathcal{E}$-Connected ontologies in its transitive closure under link references. For example, the "destinations" ontology defines a link property *providesAccommodation* to the "accommodations" ontology and a link *offersActivity* to the "activities" ontology. Since "accommodations" does not contain any link property (it is an ordinary OWL ontology) and "activities" only includes the link property *isOfferedAt* to "destinations" again, the reasoner would consider the following set of $\mathcal{E}$-Connected ontologies:

$$K_{destinations} = \{destinations, accommodations, activities\}$$

We say that $K_{destinations}$ is the $\mathcal{E}$-Connection *induced by* the *destinations* ontology. Looking at the link references between the different ontologies of the example, it is easy to see that:

$$K_{accommodations} = \{accommodations\}$$

$$K_{activities} = \{activities, destinations, accommodation\}$$

$$K_{people} = \{people, destinations, accommodations, activities\}$$

An OWL ontology O can be seen as an $\mathcal{E}$-Connected ontology which induces an $\mathcal{E}$-Connection with O as its only component.

3 Modeling with $\mathcal{E}$-Connections

$\mathcal{E}$-Connections can be used, as in the example on tourism, for integrating existing ontologies, which describe different application domains, by adding knowledge about how these domains are related to each other.

In this section, we show how to use $\mathcal{E}$-Connections the other way round, namely for decomposing a knowledge base into smaller, connected ontologies. In order to illustrate this application of $\mathcal{E}$-Connections, we have built a set of $\mathcal{E}$-Connected knowledge bases from various ontologies available on the Web and written in OWL. [4]

As an example, let us consider the ontology used in the OWL documentation: the Wine Ontology (4). This ontology describes different kinds of wines according to various criteria, like the area they are produced in, the kinds of grapes they contain, their flavor and color, etc. For example, a "Cabernet Franc Wine" is defined to be a dry, red wine, with moderate flavor and medium body and which is made with Cabernet Franc grapes [5]:

$$CabernetFranc \equiv Wine \sqcap \exists madeFromGrape.\{CabernetFrancGrape\} \sqcap \leq 1 madeFromGrape$$

$$CabernetFranc \sqsubseteq \exists hasColor.\{Red\} \sqcap \exists hasFlavor.\{Moderate\} \sqcap \exists hasBody.\{Medium\}$$

A Bordeaux is defined to be a wine produced in the Bordeaux area [6]:

$$Bordeaux \equiv Wine \sqcap \exists locatedIn.\{BordeauxRegion\}$$

Note that the Wine Ontology does not contain information about wines only, but also information about regions, wineries, colors, grapes, and so on. This

[4] The original ontologies and their corresponding $\mathcal{E}$-Connections are available online at *http://www.mindswap.org/2004/multipleOnt/FactoredOntologies*. The $\mathcal{E}$-Connected ontologies are written in an extension of OWL.

[5] In this section, instead of using the RDF/XML notation as in Section 2 we will use for clarity and brevity standard DL notation. The equivalence between this notation and abstract syntax is summarized in Tables 2 and 3.

[6] In both examples the braces represent nominals.

illustrates a general feature of OWL ontologies: although they usually refer to a core application domain, they also contain "side" information about other secondary domains.

This modeling paradigm is not only characteristic of small and medium sized ontologies, but also occurs in large, high-quality knowledge bases, written by groups of experts. A prominent example is the NCI (National Cancer Institute) ontology (13), a huge ontology describing the cancer domain. The NCI ontology is mainly focused on genes, but it also contains some information about many other domains, like professional organizations, funding, research programs, etc. [7]

In all these cases, it is more natural to represent each application domain in a different $\mathcal{E}$-Connected ontology, where link properties are used in the component KBs whenever information from a different ontology in the combination is required. The component ontologies in an $\mathcal{E}$-Connection are mostly self-contained in the sense that they only contain information about a single "topic", and are loosely coupled, since the coupling between $\mathcal{E}$-Connected ontologies can only be achieved through restrictions on link properties.

In the case of the Wine Ontology, the combined KB is composed of six ontologies, dealing with grapes, wine descriptors (color, flavor,...), regions, wineries, years, and wines respectively [8].

Figure 1 shows a visualization of the resulting $\mathcal{E}$-Connection.

In the graph, the nodes represent $\mathcal{E}$-Connected ontologies and their size is proportional to the number of entities and axioms they contain. An edge (v_1, v_2) represents the existence of a link property with source ontology v_1 and target ontology v_2.

The blue nodes in the graph represent *leaf* nodes, i.e. components that have "incoming" link properties *only*. These components are *free-standing*, in the sense that they do not use information from any other component in the $\mathcal{E}$-Connection and are written in plain OWL. On the other hand, red nodes represent the nodes with outgoing edges, which depend on the information contained in other $\mathcal{E}$-Connected ontologies.

The graph in Figure 1 is a tree. The ontology dealing with *wines* is the only one that uses information from other ontologies, which implies that wines are central to the $\mathcal{E}$-Connection. The remaining $\mathcal{E}$-Connected ontologies are

[7] The $\mathcal{E}$-Connected version of NCI is also available at *http://www.mindswap.org/2004/multipleOnt/FactoredOntologies*.
[8] In this section, we are using a version of the Wine Ontology that does not import the Food Ontology.

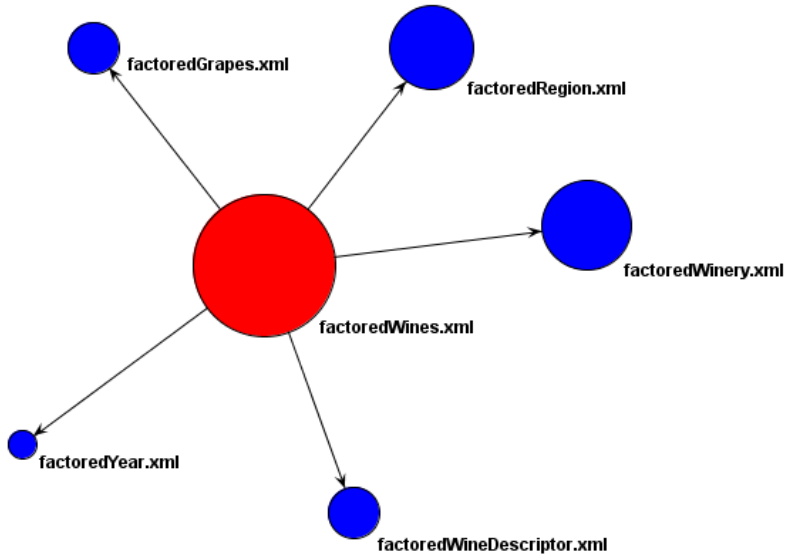

Fig. 1. The Wine Ontology as an $\mathcal{E}$-Connection

leaves in the tree, and hence represent "secondary" sub-domains. Note that, in general, the graph corresponding to an $\mathcal{E}$-Connected knowledge base can be an arbitrary directed graph.

When transforming a DL knowledge base into an $\mathcal{E}$-Connected KB, many object properties in the original ontology become link properties in the $\mathcal{E}$-Connected KB. For example, in the definition of Cabernet Franc wines, the object property *madeFromGrape* becomes a link property from the wine ontology to the grapes ontology, while *hasColor*, *hasFlavor* and *hasBody* connect the wine ontology and the wine descriptors ontology. Obviously, this effect depends on the number of $\mathcal{E}$-Connected ontologies. Since many restrictions on object properties in the original ontology become restrictions on links in the combined KB, the expressivity tends to be pushed into the link properties.

The use of $\mathcal{E}$-Connections for both integration and decomposition of OWL knowledge bases suggests a new modeling methodology, applicable to knowledge engineering with Description Logics in general, and to the Semantic Web in particular. The core idea is to keep ontologies small and disjoint and to use these ontologies as reusable units that can be combined in various ways using $\mathcal{E}$-Connections depending on the modeler's needs.

It is worth emphasizing here that $\mathcal{E}$-Connections are *not* a suitable technique for combining ontologies dealing with highly overlapping domains, which prevents its use in some important Knowledge Engineering applications, such as ontology refinement. For example, if we developed a new ontology on grapes with richer descriptions about a certain kind of grape, we would not be able to connect it to the old one using our technique. However, $\mathcal{E}$-Connections were not designed for such a purpose.

4 Tool Support

In this section, we discuss the key issues to be addressed for providing tool and application support for $\mathcal{E}$-Connections, and we describe our implementation of an $\mathcal{E}$-Connection aware infrastructure that extends the OWL-API (14) and is integrated to the ontology editor SWOOP (11). [9]

4.1 Requirements

4.1.1 Basic Functionality

A basic implementation for $\mathcal{E}$-Connections must perform similar tasks as an OWL implementation itself:

- Serializing: Producing the extended OWL concrete syntax, introduced in Section 2, from some internal representation.
- Modeling: Providing a suitable internal representation (model) for $\mathcal{E}$-Connected ontologies.
- Parsing: Building an internal representation from a document in the concrete syntax.

This basic functionality must be provided by Semantic Web high-level programming interfaces, such as the OWL-API.

Ontology browsers and editors must provide additional rendering and editing functionality. Perhaps, the most important requirement for an $\mathcal{E}$-Connections aware ontology editor is the ability to deal effectively with multiple ontologies, which implies, for example, the ability to load, save and manipulate several ontologies simultaneously, as well as the ability to easily navigate from one to another.

Ontology editors must also be able to provide browsing and editing capabilities for the extended RDF/XML and abstract syntaxes [10] and support for the simultaneous use of imports and $\mathcal{E}$-Connections.

4.1.2 Combining Imports and $\mathcal{E}$-Connections

The combined use of $\mathcal{E}$-Connections and *owl:imports* raises a number of difficulties. For example, suppose a simple case in which we have two connected

[9] SWOOP is available for download at http://www.mindswap.org/2004/SWOOP.
[10] The extended abstract syntax for $\mathcal{E}$-Connections will be presented in next section.

ontologies: the ontology O_A about "people" and the ontology O_B about "animals". Suppose that in O_A there are link properties connecting O_A to O_B and vice-versa. Assume that the modeler decides at some point that persons should be described as a subclass of animals and that the application which is using the connected ontologies relies on such a decision. Consequently, the modeler makes O_A import O_B. The following issue immediately arises: what should a tool automatically *do* in such a case?

In principle, an editor should automatically modify the ontology O_A, and leave the ontology O_B unaltered in the workspace, in order to comply with the asymmetric nature of the *owl:imports* construct. Then, the tool should transform, in O_A, all the link properties from O_A to O_B and from O_B to O_A [11] into ordinary object properties. In other words, O_A is transformed into a "plain" OWL ontology, which treats the domains of people and animals as a single one.

However, those modifications leave the ontology O_B, which is still $\mathcal{E}$-Connected to the (modified) O_A ontology, in a pretty much non-sensical state, since O_B would be $\mathcal{E}$-Connected to the "union" of O_A and itself. What should happen next? Clearly, the $\mathcal{E}$-Connection has been "broken" by the *owl:imports* statement, since, as we will discuss later, there is a violation in the restrictions on the usage of URIs within an $\mathcal{E}$-Connection.

In such a situation, "merging" O_A and O_B reveals as the most plausible solution. A *merge* would enforce that all the link properties pointing to O_B will now point to O_A (which is importing O_B) and all the link properties *from* O_B will become object properties. These operations will "disconnect" O_B from the combination.

4.2 Implementation

4.2.1 Extending the OWL-API

We have extended Manchester's OWL-API in order to provide a high-level programmatic interface for parsing, serializing and manipulating $\mathcal{E}$-Connected ontologies. Our aim has been to design a smooth extension to the API that respects its main design decisions and provides the required functionality.

The OWL-API RDF parser and internal model have been extended to deal with the new constructs. Link properties in an $\mathcal{E}$-Connected ontology are internally represented as object properties that additionally specify the URI of its target ontology. On the other hand, $\mathcal{E}$-Connected ontologies extend OWL

[11] Note that those are imported.

ontologies with functionality for retrieving link properties and foreign entities (classes, properties and individuals).

In order to manipulate the structures in the internal representation, we have provided functionality for adding/removing a foreign entity to/from an $\mathcal{E}$-Connected ontology, and to set the target ontology of a link property. The functionality for adding/removing link properties is provided by reusing the corresponding functionality for object properties.

This simple infrastructure is sufficient for building $\mathcal{E}$-Connections aware Semantic Web tools on top of the OWL-API.

4.2.2 Extending SWOOP

SWOOP (11) is a Web Ontology browser and editor, which has been especially designed for complying with the OWL nature and specifications. The main design goal has been to allow the user to create OWL ontologies rapidly and intuitively.

SWOOP assumes the use of multiple ontologies and supports this use in a number of ways. Being an open multiple ontology engineering environment, SWOOP is an ideal testbed for experimenting with $\mathcal{E}$-Connections in the Semantic Web context.

SWOOP uses the new functionality added to the OWL-API in order to provide basic rendering and editing support for $\mathcal{E}$-Connections. As in the case of the OWL-API, we have tried to keep the main UI and design decisions in SWOOP and to incorporate the required functionality with a minimal extension of the code.

The extension of the OWL-API automatically provides the functionality for loading and saving $\mathcal{E}$-Connected ontologies in the extended RDF/XML syntax presented in Section 2. When loading an $\mathcal{E}$-Connected ontology, link properties and foreign entities are distinguished in the UI from the rest of OWL entities.

The renderers have been slightly extended in order to support the new elements. The Ontology Information renderer provides useful statistics, such as the number of link properties, foreign classes, properties and individuals in an $\mathcal{E}$-Connected ontology. The concise format entity renderer has been extended to display properly link properties and foreign entities. In the case of link properties, the renderer pane shows its *target* ontology, while in the case of foreign entities the pane shows the (foreign) ontology in which the entity has been defined. All these elements are hyperlinked; thus, for example, if the user clicks on the URI of a foreign ontology, the ontology will be loaded in the workspace. The idea is to provide easy and intuitive navigation through the

ontologies in an $\mathcal{E}$-Connection. Finally, the Abstract Syntax entity renderer displays the extended OWL abstract syntax that will be presented in next section, while the RDF/XML entity renderer provides the extended RDF/XML code for the entity, as presented in Section 2.

SWOOP provides basic tool support for creating and editing $\mathcal{E}$-Connected ontologies. Link properties can be added to the ontology and it is possible to define restrictions on them. Incorrect editing is prevented when possible: link properties cannot be made transitive or symmetric and restrictions on link properties can only be applied to classes/individuals in its *target* ontology.

Finally, SWOOP provides a graph layout for visualizing the connections between components in an $\mathcal{E}$-Connection. Given a selected ontology in the workspace, SWOOP allows to display its induced $\mathcal{E}$-Connection. Figure 1 shows the graph corresponding to the Wine Ontology.

5 An Extension of OWL-DL

5.1 Abstract Syntax

The syntax presented in this section is an extension of the normative OWL abstract syntax, as described in the OWL abstract syntax and semantics recommendation (3), and we use the same Extended BNF syntax as in the normative document.

An $\mathcal{E}$-Connected ontology K contains a sequence of annotations, axioms and facts. Annotations, as in OWL, can be used to record authorship and other information associated with the ontology, including imports.

E-ConnectedOntology :: =
'E-ConnectedOntology('[**ontologyID**] { **directive** } ')'

directive :: = 'Annotation(' **ontologyPropertyID ontologyID** ')'
| 'Annotation(' **annotationPropertyID URIreference** ')'
| 'Annotation(' **annotationPropertyID dataLiteral** ')'
| 'Annotation(' **annotationPropertyID individual** ')'
| **axiom**
| **fact**

$\mathcal{E}$-Connected ontologies are referenced using a URI. $\mathcal{E}$-Connected ontologies contain information about the same kind of entities as OWL ontologies (classes, object properties, etc.), but they also contain information about link properties. Link properties are also denoted by URI references.

linkID :: = URIreference

In order to ensure the separation of vocabularies, a URI-reference cannot be both a linkID, an individual-valued property ID, a datatype property ID, an annotation property ID or an ontology property ID in an $\mathcal{E}$-Connected ontology. Intuitively, while individual-valued properties relate individuals to other individuals in the same ontology, link properties relate individuals corresponding to different interpretation domains. Thus, link properties act as a bridge between different ontologies, which remain separate, and keep their own identity.

An OWL ontology in the abstract syntax contains a sequence of axioms and facts. In order to provide support for $\mathcal{E}$-Connections on the Semantic Web, we propose a syntactic and semantic extension of OWL-DL with new kinds of axioms and facts. Every OWL ontology can be seen as an $\mathcal{E}$-Connected ontology in which no link properties have been declared.

5.1.1 Axioms

In a $\mathcal{E}$-Connected ontology, link properties can be defined using the following axiom:

axiom ::= 'Link(' **linkID**['Deprecated'] { **annotation** }
'foreignOntology(' **OntologyID**')'
{ 'super(' linkID ')' }
{ 'domain(' **description** ')' }
{ 'range(' **foreignDescription** ')' }
['inverseOf(' **linkID** ')']
['Functional' | 'InverseFunctional']

Link properties used in an abstract syntax ontology *must* be declared, and hence need an axiom. A link property cannot be declared twice as referring to different ontologies.

Link properties can be made functional or inverse functional and can be given global domains and ranges. As opposed to object properties, link properties cannot be made transitive or symmetric.

Link properties can be equivalent to or sub-properties of others. Of course, in order for these axioms to model useful information, the link properties that are related through equivalence or subsumption should refer to the same foreign ontology.

axiom ::= 'EquivalentProperties(' **linkID linkID** { **linkID** } ')'

| 'SubPropertyOf(' **linkID linkID** ')'

In $\mathcal{E}$-Connections the coupling between the ontologies is achieved through restrictions on link properties. As in OWL, universal (*allValuesFrom*), existential (*someValuesFrom*) and value (*hasValue*) restrictions can be defined.

restriction::= 'Restriction('**linkID**
linkRestrictionComponent { **linkRestrictionComponent** } ')'

linkRestrictionComponent::= 'allValuesFrom(' **foreignDescription** ')'
| 'someValuesFrom(' **foreignDescription** ')'
| 'value(ForeignIndividual(' **individualID** '))'
| **cardinality**

Range axioms and restrictions on link properties are referred to foreign class descriptions. Foreign classes are classes that, though used in a certain $\mathcal{E}$-Connected ontology, correspond to a different ontology in the combined knowledge base. If a foreign description is used in a range axiom or in a restriction corresponding to a link property, it will always be interpreted in the domain of the target ontology of the link property.

foreignDescription ::= 'ForeignClass(' **description** ')'

5.1.2 Facts

Our proposal extends the OWL-DL facts by adding the following production rule to the normative OWL abstract syntax:

value ::= 'value(' ForeignIndividual(**linkID** individualID ')'

These facts allow to instantiate link properties with named individuals.

5.2 Direct Model-Theoretic Semantics

This section provides a model-theoretic semantics to $\mathcal{E}$-Connected ontologies written in the abstract syntax.

As in OWL, the definition of the semantics starts with the notion of a *combined vocabulary*.

Definition 1 *A **combined OWL vocabulary** V consists of a set V_L of literals and the following sets of URI references: For $i = 1, ..., n$, V_{C_i} are sets of class names, each of which contains owl:Thing and owl:Nothing; V_{I_i} are sets*

of individual names; V_{DP_i}, V_{IP_i} and V_{AP_i} are sets of datatype, object and annotation property names respectively, where each V_{AP_i} contains owl:versionInfo, rdfs:label, rdfs:comment, isDefinedBy, seeAlso; V_D is the set of datatype names, which also contains the URI references for the built-in OWL datatypes and rdfs:Literal; V_O the set of ontology names and V_{OP} the set of URI references for the built-in ontology properties; finally, for $i, j = 1, ..., n$ with $i \neq j$, $\mathcal{E}_{ij}$ are sets of URI references denoting link properties.

In any vocabulary, the $(V_{C_i} - \{owl : Thing, owl : Nothing\})$ are pair-wise disjoint, and disjoint with V_D. Also, $\forall i, j = 1, ..., n; i \neq j$, the $V_{DP_i}, V_{IP_i}, (V_{AP_i} - \{owl : versionInfo, rdfs : label, rdfs : comment\}), V_{OP}, \mathcal{E}_{ij}$ are all pair-wise disjoint.

Given an $\mathcal{E}$-Connected ontology, the vocabulary *must* include all the URI references and literals utilized in each of the ontologies, as well as those used in ontologies that are imported or referenced through a link property by any of the component ontologies, but can contain other URI references and literals as well.

As in OWL, a datatype d is characterized by a lexical space, $L(d)$, which is a set of Unicode strings; a value space, $V(d)$; and a total mapping $L_2V(d)$ from the lexical space to the value space.

Definition 2 *A **datatype map** D is a partial mapping from URI references to datatypes that maps xsd:string and xsd:integer to the appropriate XML Schema datatypes.*

The model-theoretic semantics is provided by the notion of a *combined interpretation*.

Definition 3 *Let D be a datatype map. A **combined OWL interpretation** with respect to D with combined vocabulary V is a tuple of the form:*

$$I = (R, (R_i)_{1 \leq i \leq n}, (EC_i)_{1 \leq i \leq n}, (ER_i)_{1 \leq i \leq n}, L, (S_i)_{1 \leq i \leq n}, ED, LV, N),$$

where (with $\mathcal{P}$ being the powerset operator):

- *R the set of resources of the interpretation is a non-empty set*
- *$LV \subset R$, the set of literal values of I, contains the set of Unicode strings, the set of pairs of Unicode strings and language tags, and the value spaces for each datatype in D*
- *$R_i = O_i \cup LV$, where O_i is non-empty, disjoint from LV and disjoint from $O_j, \forall j = 1, ..., n; i \neq j$*
- *$EC_i : V_{C_i} \rightarrow \mathcal{P}(O_i)$*
- *$ED : V_D \rightarrow \mathcal{P}(LV)$*
- *$ER_i : V_{IP_i} \rightarrow \mathcal{P}(O_i \times O_i)$*

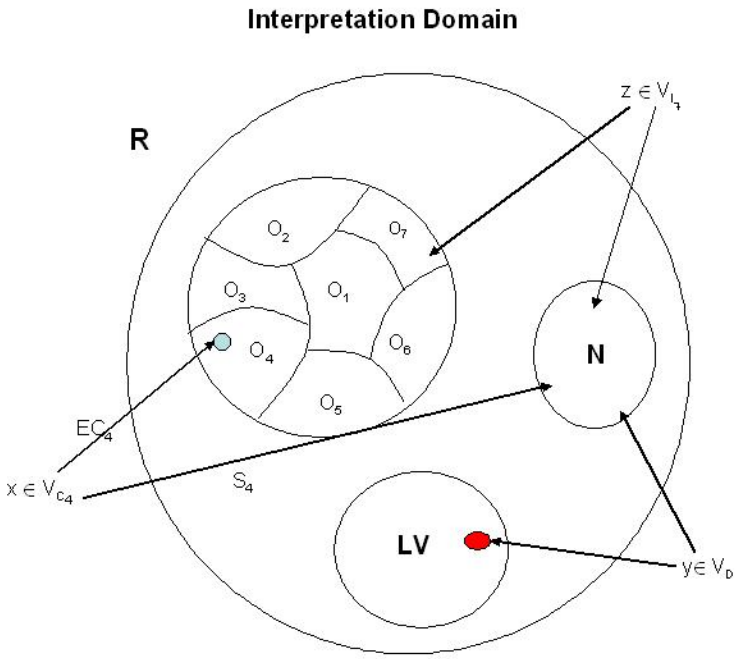

Fig. 2. A Combined Interpretation

- $ER_i : V_{DP_i} \rightarrow \mathcal{P}(O_i \times LV)$
- $ER_i : \mathcal{E}_{ij} \rightarrow \mathcal{P}(O_i \times O_j)$
- $L : TL \rightarrow LV$, *where* TL *is the set of typed literals in* V_L
- $ER_i : V_{AP_i} \rightarrow \mathcal{P}(R_i \times R_i)$
- $ER_i : V_{OP} \rightarrow \mathcal{P}(R_i \times R_i)$
- $S_i : V_{I_i} \rightarrow O_i$
- $S_i : V_{I_i} \cup V_{C_i} \cup V_D \cup V_{DP_i} \cup V_{IP_i} \cup V_{AP_i} \cup \mathcal{E}_{ij} \cup \{$ *owl:Ontology, owl:DeprecatedClass, owl:DeprecatedProperty* $\} \rightarrow N$, *where* $N \subset R$ [12] *is disjoint with* LV *and with each of the* $O_i, \forall i = 1, ..., n$
- $EC_i(owl : Thing) = O_i \subseteq R$
- $EC_i(owl : Nothing) = \emptyset$
- $EC_i(rdfs : Literal) = LV$
- *If* $D(d') = d$ *then* $ED(d') = V(d)$
- *If* $D(d') = d$, *then* $L(\text{`}v'\text{`}\hat{}\hat{}d') = L_2V(d)(v)$
- *If* $D(d') = d$ *and* $v \notin L(d)$, *then* $L(\text{`}v'\text{`}\hat{}\hat{}d') \in R - LV$

In a combined OWL interpretation, the functions $(EC_i)_{1 \leq i \leq n}$ provide logical meaning for URI references used as classes in the $\mathcal{E}$-Connected ontology K_i. The functions $(ER_i)_{1 \leq i \leq n}$ assign logical meaning to the URIs in the ontologies $(K_i)_{1 \leq i \leq n}$ that are used as OWL properties or links [13]. In a combined interpretation the abstract interpretation domain is partitioned into n disjoint parts,

[12] Vocabulary Names.
[13] As in OWL, the property *rdf:type* is added to the annotation properties in order to provide meaning for deprecation.

Abstract Syntax	Interpretation (value of EC_i)
restriction(l allValuesFrom(ForeignClass(c)))	$\{x \in O_i \mid (x,y) \in ER_i(l) \Rightarrow y \in EC_j(c)\}$
restriction(l someValuesFrom(ForeignClass(c)))	$\{x \in O_i \mid (x,y) \in ER_i(l) \wedge y \in EC_j(c)\}$
restriction(l value(ForeignIndividual(id)))	$\{x \in O_i \mid (x, S_j(id)) \in ER_i(l)\}$
restriction(l minCardinality(n))	$\{x \in O_i \mid \#(\{y \in O_j : (x,y) \in ER_i(l)\}) \geq n\}$
restriction(l maxCardinality(n))	$\{x \in O_i \mid \#(\{y \in O_j : (x,y) \in ER_i(l)\}) \leq n\}$
restriction(l Cardinality(n))	$\{x \in O_i \mid \#(\{y \in O_j : (x,y) \in ER_i(l)\}) = n\}$

Table 1 Extension of EC_i

each of which corresponding to a different component ontology. Link properties are interpreted through $(ER_i)_{1 \leq i \leq n}$ as pairs of elements corresponding to different parts of the abstract logical domain.

The function L provides meaning for typed literals and the function ED to the datatypes used in the $\mathcal{E}$-Connection. Note that, as opposed to the abstract interpretation domain, the domain in which datatypes and literals are interpreted is not partitioned.

The functions $(S_i)_{1 \leq i \leq n}$ provide meaning to OWL individuals. Analogously to OWL, these functions are extended to plain literals in V_L by mapping them onto themselves. Note that, if the same literal "l" is used in different component ontologies, say i, j, the functions S_i, S_j will map it to the same value in LV, i.e. $S_i(l) = S_j(l) = l \in LV$.

The functions EC_i are extended to class descriptions, individuals and annotations an in the OWL specification (3); the additional constructs are interpreted according to Table 1, where l is a link property declared in K_i with foreign ontology K_j.

A combined OWL interpretation I satisfies axioms and facts as given in the OWL specifications (3) and in Table 2.

We now define precisely the notion of the $\mathcal{E}$-Connection induced by an ontology, introduced informally in Section 2.3.

Definition 4 *Let K_1 be an $\mathcal{E}$-Connected ontology. The $\mathcal{E}$-Connection* ***induced by*** *of K_1 is defined to be the following set of $\mathcal{E}$-Connected ontologies:*

$$K = (K_1, ..., K_n) = clos(K_1)$$

where the set $clos(K_1)$ is inductively defined as follows:

Directive	Conditions on Interpretations
Link(l [Deprecated]	$[(S_i(l), S(\text{ owl:DeprecatedProperty })) \in ER_i(\text{rdf:type})]$
foreignOntology(Ont_j)	$ER_i(l) \subseteq O_i \times O_j$
annotation(p_1 o_1) ...	$S_i(l) \in EC_i(annotation(p_1\ o_1))...$
... annotation(p_k o_k)	$...S_i(l) \in EC_i(annotation(p_k\ o_k))$
super(s_1) ... super(s_n))	$ER_i(l) \subseteq ER_i(s_1) \cap ... \cap ER_i(s_n)$
domain(d_1)... domain(d_n)	$ER_i(l) \subseteq EC_i(d_1) \times O_j \cap ... \cap EC_i(d_n) \times O_j$
range(r_1) ... range(r_n)	$ER_i(l) \subseteq O_i \times EC_j(r_1) \cap ... \cap O_i \times EC_j(r_n)$
[inverseOf(m)]	$ER_i(l) = (ER_j(m))^-$
[Functional]	$\forall x \in O_i\ \forall y, z \in O_j, (ER_i(l))(x, y)$
	$\wedge (ER_i(l))(x, z) \rightarrow y = z$
[InverseFunctional]	$\forall y, z \in O_i, \forall x \in O_j, (ER_i(l))(y, x)$
	$\wedge (ER_i(l))(z, x) \rightarrow y = z$
EquivalentProperties($l_1...l_n$)	$ER_i(l_j) = ER_i(l_k) \forall 1 \leq j \leq k \leq n$
SubpropertyOf(l_1, l_2)	$ER_i(l_1) \subseteq ER_i(l_2)$

Table 2 Interpretation of Axioms and Facts

- $K_1 \in clos(K_1)$.
- *If K' belongs to $clos(K_1)$ and there is a link property with source K' and target K'', then $K'' \in clos(K_1)$.*

Note that an $\mathcal{E}$-Connected ontology K_1 is an "ordinary" OWL ontology, as specified in the normative documents if and only if the $\mathcal{E}$-Connection induced by K_1 is precisely itself, i.e. $K = (K_1)$.

Since a URI can be used in an $\mathcal{E}$-Connected ontology either as *local* or *foreign*, we need to provide a formal distinction between both cases. For such a purpose, we introduce the notion of a URI to belong to an $\mathcal{E}$-Connected ontology as follows:

Definition 5 *Let $K = (K_1, ..., K_n)$ be the $\mathcal{E}$-Connection induced by K_1. We say that a URI reference u* ***belongs to*** *the $\mathcal{E}$-Connected ontology K_i if either of the following conditions holds:*

- *It is used in K_i or in an ontology imported by K_i, but not in the context of a restriction on a link property.*
- *It is used in $K_j, j \neq i$ in the context of a restriction on a link property with foreign ontology K_i.*

For example, assume that the URI *foo:Publication* is used in an $\mathcal{E}$-Connected

ontology O_1 in the context of the following restriction: restriction(l Foreign-Class (foo:Publication)), where l is defined to be a link property with foreign ontology O_2; then, the URI would *belong to* the ontology O_2 and would *necessarily* need to be declared as a class in O_2.

The semantics of an $\mathcal{E}$-Connection is given as follows:

Definition 6 *Let D be a datatype map, K_1 be an $\mathcal{E}$-Connected ontology, and $K = (K_1, ..., K_n)$ the $\mathcal{E}$-Connection induced by K_1 . A combined OWL interpretation I with respect to D with vocabulary V, **satisfies** K_1 (denoted by $I \models K_1$) iff:*

*(1) Each URI reference **belonging to** K_i, used as a classID (datatypeID, individualID, data-valued property ID, annotation property ID, annotation ID, ontology ID) belongs to a* single *component ontology K_i and is contained in $V_{C_i}(V_D, V_{I_i}, V_{DP_i}, V_{IP_i}, V_{AP_i}, V_O$, respectively).*
(2) Each literal in K_i belongs to V_L.
(3) Each URI reference in K_i used as a linkID, with foreign ontology K_j, is in $\mathcal{E}_{ij}$.
(4) I satisfies each directive in K_i, except for ontology annotations.
(5) There is, for all $i = 1, ..., n$ some $o_i \in N \subset R$ with $(o_i, S_i(owl : Ontology)) \in ER_i(rdf : type)$ such that for each ontology annotation of the form $Annotation(p\ v)$, $(o_i, S_i(v)) \in ER_i(p)$ and if the component ontology K_i has name n_i, then $S_i(n_i) = o_i$.
(6) I satisfies each $\mathcal{E}$-Connected ontology in K.
(7) I satisfies each ontology mentioned in an owl:imports annotation directive of any K_i.

At this point, it is worth discussing the semantics we have provided to URIs within an $\mathcal{E}$-Connection.

The meaning of names is a contentious issue in the Semantic Web. Numerous proposals have been given for how to provide meaning for names in the Semantic Web, ranging from a strict localized model-theoretic semantics to proposals for a unified single meaning. Certainly, the latter was the original goal of URIs, as "global" identifiers. However, currently, the meaning of a name in RDF and hence in OWL is relative to a particular RDF graph (15). In other words, the meaning of the same URI in *other* documents is not considered *at all*. The only way that the OWL standards provide in order to take into account the meaning of a URI in a different ontology is to *import* that ontology. When an ontology imports another one, identical URIs are *merged*; otherwise, the meaning of a URI is entirely "local" to an RDF/OWL document.

In the framework of $\mathcal{E}$-Connections, we provide a *stronger* meaning to URIs. Roughly, we prevent the use of the same global name (URI reference) to denote *different* things within an $\mathcal{E}$-Connection. Suppose we use the same

URI to denote an individual in two different ontologies. For example, assume we use the URI *http://www.mindswap.org/bob* to denote an individual in an ontology about people and in an ontology about "pets" and suppose that we consider an $\mathcal{E}$-Connection which includes both ontologies. It is a fundamental assumption in the $\mathcal{E}$-Connections formalism that the (interpretation) domains of the component ontologies are disjoint. Hence, if the URI *http://www.mindswap.org/bob* is used as a name for an individual in both the "pets" and "people" ontology, it *necessarily* refers to two different entities. In such a case, an $\mathcal{E}$-Connections aware reasoner should detect an inconsistency. Thus, we are extending the meaning of a URI from a single document to the context of its induced $\mathcal{E}$-Connection.

Recall also that, although each ontology within an $\mathcal{E}$-Connection is interpreted in a different domain, datatypes in every ontology are interpreted in the *same* domain. As opposed to the case of object domains, there is no reason to partition the datatype domain. Note that the original $\mathcal{E}$-Connections framework as presented in (8) does not consider datatype theories, nor we considered datatypes in the reasoning algorithms presented in (16). However, our approach in this paper concerning datatypes does not affect the results in (8) and allows for a straightforward extension of the algorithms in (16).

The main reasoning services are, as in OWL, consistency and entailment.

Definition 7 *An $\mathcal{E}$-Connected ontology K_1 is **consistent** with respect to a datatype map D (denoted by $I \models_D K$ iff there is some combined interpretation I with respect to D such that I satisfies K_1.*

Definition 8 *An $\mathcal{E}$-Connected ontology K_1 **entails** an $\mathcal{E}$-Connected ontology K_2 with respect to a datatype map D, denoted by $K_1 \models_D K_2$ iff every combined interpretation that satisfies K_1 with respect to D also satisfies K_2 with respect to D.*

The following results are a straightforward consequence of the definitions of entailment and satisfaction under a combined interpretation:

Consequence 1 *Let K_1 be an $\mathcal{E}$-Connected ontology and $K = (K_1, ..., K_n)$ be the $\mathcal{E}$-Connection induced by K_1. Then $K_1 \models_D K_j,\ \forall j = 1, ..., n$.*

Consequence 2 *The $\mathcal{E}$-Connection $K = (K_1, ..., K_n)$ induced by K_1 entails the $\mathcal{E}$-Connection $O = (O_1, ..., O_m)$ induced by O_1 with respect to a datatype map D ($K \models_D O$) iff $K_1 \models_D O_1$.*

These results show that a $\mathcal{E}$-Connection is identified in a Semantic Web context by its "generating" $\mathcal{E}$-Connected ontology.

Construct Name	DL Syntax	OWL Syntax	Logic
Atomic Concept	A	A(URI reference)	
Universal Concept	$\top$	owl:Thing	
Atomic Role	R	R (URI reference)	
Conjunction	$C \sqcap D$	intersectionOf(C,D)	$\mathcal{S}$
Disjunction	$C \sqcup D$	unionOf(C,D)	
Negation	$\neg C$	ComplementOf(C)	
Existential Restriction	$\exists R.C$	restriction(R someValuesFrom(C))	
Value Restriction	$\forall R.C$	restriction(R allValuesFrom(C))	
Transitive Role	$Trans(R)$	ObjectProperty(R [Transitive])	
Role Hierarchy	$R \sqsubseteq S$	subPropertyOf(R,S)	$\mathcal{H}$
Inverse Role	$S = R^-$	ObjectProperty(S [inverseOf(R)])	$\mathcal{I}$
Nominals	$\{o_1, ..., o_n\}$	OneOf($o_1, ..., o_n$)	O
Functional Role	$Funct(R)$	ObjectProperty(R [Functional])	$\mathcal{F}$
Functional	$\geq 2R$	restriction(R minCardinality(1))	
Number Restrictions	$\leq 1R$	restriction(R maxCardinality(1))	
Unqualified	$\geq nR$	restriction(R minCardinality(n))	$\mathcal{N}$
Number Restrictions	$\leq nR$	restriction(R maxCardinality(n))	

Table 3 The $\mathcal{SH}$ family of Description Logics

6 Reasoning

6.1 Reasoning in OWL

OWL-DL and OWL-Lite can be seen as expressive Description Logics, with an ontology being equivalent to a Description Logics knowledge base. Among the myriad of very expressive DLs, the $\mathcal{SH}$ family of logics plays a prominent role (17). All modern, highly optimized DL reasoners, such as FaCT (18), RACER (19) and Pellet (12) have been designed for these logics.

In order to obtain a suitable balance between computational properties and expressivity, the design of the DL-based species of OWL has been grounded on the $\mathcal{SH}$ family of logics. OWL-Lite corresponds to $\mathcal{SHIF}(\mathcal{D})$, while OWL-DL corresponds to $\mathcal{SHOIN}(\mathcal{D})$.

The first algorithm for the logic $\mathcal{SH}$ was presented in (20). The extension for

$\mathcal{SHIF}$ was presented in (21), and qualified number restrictions (an extension of the number restrictions used in OWL) were introduced for the logic $\mathcal{SHIQ}$ in (17). Nominals and datatypes were presented for the logic $\mathcal{SHOQ}(\mathcal{D})$ in (22). In (23) it was proved using the tableau systems formalism that satisfiability in $\mathcal{SHIO}$ is can be decided with a tableau algorithm. Finally, the design of a reasoning procedure for $\mathcal{SHOIQ}$ has been achieved only very recently (24).

6.2 A Family of $\mathcal{E}$-Connection Laguages

There are two ways to obtain new $\mathcal{E}$-Connection languages. The first possibility is to vary the set of component logics; the second would be to change the logic of the link properties, i.e., to vary the set of operators that can be applied on link properties. Different choices in the component logics and in the logic of the link properties will yield, in general, different combination languages, with different expressivity and computational properties.

A family of $\mathcal{E}$-Connection languages can be specified by fixing the component logics and varying the logic of the link properties. In this paper, we have tacitly specified, in the abstract syntax and semantics section the $\mathcal{E}$-Connection language $\mathcal{C}^{\mathcal{E}}_{\mathcal{IHN}}(\mathcal{SHOIN}(\mathcal{D}))$, i.e., the $\mathcal{E}$-Connection language that allows the use all the expressivity of $\mathcal{SHOIN}(\mathcal{D})$, and hence of OWL-DL, in the component ontologies, and the use of inverse link properties, link hierarchies and *someValuesFrom*, *allValuesFrom*, *hasValue* and Cardinality restrictions on link properties.

It is not hard to see that the first requirement for reasoning on an $\mathcal{E}$-Connection language is the ability to reason independently on each of its component logics. Our aim is to show that it is possible to design and implement practical tableau-based algorithms for $\mathcal{E}$-Connections on top of existing DL reasoners. Therefore, in this paper, we will focus on the family of combination languages having as component logics the three most prominent fragments of $\mathcal{SHOIN}(\mathcal{D})$ that have been implemented in DL reasoners, namely $\mathcal{SHIN}(\mathcal{D})$, $\mathcal{SHON}(\mathcal{D})$ and $\mathcal{SHIO}(\mathcal{D})$ and investigate their expressiveness, usefulness, and computational properties.

Once the component logics have been identified, it remains to investigate the expressivity that should be allowed on link properties. The language $\mathcal{C}^{\mathcal{E}}_{\mathcal{IHN}}(\mathcal{SHIN}(\mathcal{D}), \mathcal{SHON}(\mathcal{D}), \mathcal{SHIO}(\mathcal{D}))$ is the most expressive fragment of $\mathcal{C}^{\mathcal{E}}_{\mathcal{IHN}}(\mathcal{SHION}(\mathcal{D}))$ that can be obtained using as component logics $\mathcal{SHIN}(\mathcal{D})$, $\mathcal{SHON}(\mathcal{D})$ and $\mathcal{SHIO}(\mathcal{D})$. In this language, each of the component ontologies can be written *either* in $\mathcal{SHIN}(\mathcal{D})$, *or* in $\mathcal{SHON}(\mathcal{D})$, *or* in $\mathcal{SHIO}(\mathcal{D})$, and the full expressivity on link properties is allowed.

Construct Name	Syntax	Naming Scheme
Atomic Link	$p \in \mathcal{E}_{ij}$	
Existential Restriction	$\exists p.Z$	$\mathcal{C}(\mathcal{SHIN}(\mathcal{D}), \mathcal{SHON}(\mathcal{D}), \mathcal{SHIO}(\mathcal{D}))$
Universal Restriction	$\forall p.Z$	
Atleast Number Restriction	$\geq np$	$\mathcal{N}$
Atmost Number Restriction	$\leq np$	
Link Inversion	p^-	$\mathcal{I}$
Funct. Link Axiom	$Funct(p)$	$\mathcal{F}$
Funct. Number Restrictions	$\geq 2p, \leq 1p$	
Link Inclusion Axiom	$p \sqsubseteq q$	$\mathcal{H}$

Table 4 $\mathcal{E}$-Connection Languages

However, due to the presence of inverses in this language, cardinality restrictions on link properties allow to transfer nominals in its full generality from one component to another (8), which would invalidate the separation of domains and hence "break" the $\mathcal{E}$-Connection. [14] .

However, it is worth emphasizing here that, in many special cases, we can still process $\mathcal{C}^{\mathcal{E}}_{\mathcal{IHN}}(\mathcal{SHIN}(\mathcal{D}), \mathcal{SHON}(\mathcal{D}), \mathcal{SHIO}(\mathcal{D}))$ combined ontologies, even if number restrictions and inverses on link properties, as well as nominals, are used in the combination.

A $\mathcal{C}^{\mathcal{E}}_{\mathcal{IHN}}(\mathcal{SHIN}(\mathcal{D}), \mathcal{SHON}(\mathcal{D}), \mathcal{SHIO}(\mathcal{D}))$ $\mathcal{E}$-Connected KB $K = (K_1, ..., K_n)$ can be processed as long as, for every possible pair of component ontologies K_i, K_j for $i, j \in \{1, ..., n\}, i \neq j$, the following conditions *do not hold simultaneously*:

(1) Any of K_i, K_j contains nominals.
(2) An inverse of a link property from K_i to K_j or vice-versa is used.
(3) Number restrictions on links from K_i to K_j or vice-versa are used.

Of course, in order to be able to handle arbitrary combined ontologies, we need to further restrict the expressivity allowed on link properties.

In the sections that follow, we will discuss how the two obvious fragments of $\mathcal{C}^{\mathcal{E}}_{\mathcal{IHN}}(\mathcal{SHIN}(\mathcal{D}), \mathcal{SHON}(\mathcal{D}), \mathcal{SHIO}(\mathcal{D}))$ not containing inverses and cardinality restrictions on link properties simultaneously can be handled algorithmically.

[14] Obviously, if none of the component logics contained nominals, it would be safe to use the full expressivity on the link properties.

So far, we have discussed which $\mathcal{E}$-Connection languages we can handle algorithmically, but we have not yet explained *how* the reasoning is actually performed and implemented. In this section, we provide a general intuition on how the reasoning algorithms we have developed for $\mathcal{E}$-Connections work. We will not include here a detailed formal presentation of the algorithms, nor the proofs of correctness and completeness; we refer the interested reader to (16) for a detailed discussion.

Modern DL reasoners, like FaCT, RACER and Pellet implement the tableaux method (25) for solving the main reasoning tasks in expressive Description Logics.

Tableau algorithms are focused on satisfiability; other reasoning problems, like subsumption, or entailment (the main inference problem in OWL) can be solved by first reducing them to satisfiability (26).

In order to check satisfiability of a given concept C_0 w.r.t. a knowledge base Σ, a tableau algorithm tries to construct a common model of C_0 and Σ. If it succeeds, then the algorithm determines that C_0 is satisfiable w.r.t. Σ, and unsatisfiable otherwise.

The main elements that specify a tableau algorithm are (27):

- An underlying data structure, called the completion graph.
- A set of expansion rules.
- A blocking condition, for ensuring termination.
- A set of clash-triggers, to detect logical contradictions (clashes).

Completion graphs are finite, labeled, directed graphs, which roughly correspond to abstractions of possible models for C and Σ. Depending on the DL under consideration, completion graphs can be restricted to trees (for example, in $\mathcal{SHIQ}$), forests (in $\mathcal{SHOQ}$ and $\mathcal{SHIO}$) or to directed acyclic graphs ($\mathcal{SHOIN}$).

Each node and edge in a completion graph $\mathcal{G}$ is labeled with a set of concepts and a set of roles respectively. To decide the satisfiability of C w.r.t. Σ, the algorithm generates an initial graph $\mathcal{G}$, constructed from C and Σ and repeatedly applies the expansion rules until a clash (i.e. a contradiction) is detected in the label of a node, or until a clash-free graph is found to which no more rules are applicable. The application of a rule may add new concepts to the label of a node, trigger the generation of a new node or cause two different nodes to merge.

Tableau algorithms for expressive DLs are *non-deterministic* in the sense that there might exist completion rules that yield more than one possible outcome. A tableau algorithm will return "satisfiable" iff there exists at least one way to apply the non-deterministic rules such that a clash-free graph is obtained, to which no rules are applicable.

Termination is guaranteed through *blocking*: halting the expansion process when a "cycle" is detected (25). When the algorithm detects that a path in the graph will be expanded forever without encountering a contradiction, then the application of the generating rules is blocked, so that no new nodes will be added to that path. There are different kinds of blocking conditions, depending on the presence of inverses and number restrictions in the logic.

The basic strategy to extend a DL tableau algorithm with $\mathcal{E}$-Connections support is based on "coloring" the completion graph [15]. Nodes of different "colors", or sorts, correspond to different domains (ontologies).

The presence of *someValuesFrom* and *minCardinality* restrictions on a node label trigger the generation of a successor node of a different "color". The presence of different kinds of nodes in a the graph has several consequences in the way the tableau algorithm works:

- A node may only be blocked by an ancestor node with the same color, and the blocking condition applied to those nodes depends on the logic of the corresponding component ontologies. This implies, for example, that if a certain component ontology does not contain inverse object properties, we can apply subset blocking to its corresponding nodes in the tableau expansion even if *other* component ontologies do contain inverses.
- If a node x is a successor of a node y with a different color, then a special blocking condition is applied to x. Such a blocking condition *only* depends on the logic of the link properties, and not on the logic of the component ontology to which x corresponds. Intuitively, if inverses on link properties are present, then equality blocking is applied; otherwise, subset blocking suffices.
- In Description Logics, a TBox (the part of the ontology that does not contain facts) can be made equivalent to a single class. Such a concept is then added to the label of all the nodes in the graph during the execution of the algorithm. With $\mathcal{E}$-Connections, each component ontology TBox is made equivalent to a different class, and each of those classes is added only to the labels of nodes corresponding to that component ontology.
- The presence of *maxCardinality restrictions* and nominals (has-Value, oneOf) in node labels may cause the merge of two nodes in the graph. Obviously,

[15] Please, note that our problem has no relation with the graph coloring problem. We just use the term "color" metaphorically to distinguish between different kinds of nodes.

only nodes of the same color can ever be merged.

When implementing tableau algorithms for $\mathcal{E}$-Connections as an extension of an OWL reasoner, all these issues have to be thoroughly considered.

6.4 Implementation in an OWL reasoner

We have implemented the tableau algorithms for the $\mathcal{E}$-Connection languages $\mathcal{C}^{\mathcal{E}}_{\mathcal{IH}}(\mathcal{SHIN}(\mathcal{D}), \mathcal{SHON}(\mathcal{D}), \mathcal{SHIO}(\mathcal{D}))$ and $\mathcal{C}^{\mathcal{E}}_{\mathcal{HN}}(\mathcal{SHIN}(\mathcal{D}), \mathcal{SHON}(\mathcal{D}), \mathcal{SHIO}(\mathcal{D}))$ in the OWL reasoner Pellet.

Pellet is a sound and complete tableau reasoner for the Description Logics $\mathcal{SHIN}(\mathcal{D})$, $\mathcal{SHON}(\mathcal{D})$ and $\mathcal{SHIO}(\mathcal{D})$ (with ABoxes). Pellet implements the usual suite of optimizations, including lazy unfolding, absorption, dependency directed backjumping, and semantic branching. It provides support for handling conjunctive ABox queries and incorporates datatype reasoning for the built-in primitive XML Schema datatypes. Pellet is implemented in pure Java and available as open source software.

Pellet has been extended to process $\mathcal{E}$-Connected ontologies written in the RDF/XML syntax presented in the first part of this paper. Reasoning support for $\mathcal{E}$-Connections has been integrated as a true extension of Pellet, since reasoning with a single knowledge base can be seen as a particular case of $\mathcal{E}$-Connections where no link properties are present. An $\mathcal{E}$-Connected knowledge base has been implemented as a collection of TBoxes and RBoxes, indexed by the ontology they correspond to. During parsing, each class, object property, datatype property and link property is added to its corresponding component, and after parsing each component of the KB is pre-processed separately.

When performing a satisfiability test, the nodes in the tableau expansion are also labeled with the ontology they refer to. Links are implemented as object properties of a special kind, since they indicate the name of the foreign ontology they point to. When the generating link rules are applied, the ontology label of the new nodes is set to the foreign ontology of the link property involved in the rule application. The distinction between different kinds of nodes also implies that the rules will only merge nodes belonging to the same ontology, and the class names in labels will be replaced by their definition in the corresponding ontology.

Finally, blocking distinguishes between nodes which are generated as link successors and nodes created as successors of common roles, since different conditions apply.

When a KB is represented using $\mathcal{E}$-Connections, we can use a set of *optimiza-*

tion techniques that would not be applicable if the KB had been represented monolithically using a single KB in OWL. All these techniques take advantage of the partitioning of the domains in order to reduce the computational cost:

- **Detection of obvious non-subsumptioms**: Detecting non-subsumptions is hard for tableaux algorithms (28). Typically, when computing the classification hierarchy of a DL ontology, many subsumption tests that are performed at each node are very likely to fail. These unnecessary tests can be computationally costly and also very repetitive, and hence they affect significantly the performance of a DL reasoner. This problem is usually dealt with using caching, an optimization technique that allows to prove non-subsumptions by using cached results from previous satisfiability tests.

  In an $\mathcal{E}$-Connection, many non-subsumptions become obvious, since a subsumption test $A \sqsubseteq B$ involving two classes A, B belonging to different ontologies in the $\mathcal{E}$-Connection will necessarily fail. However, if these classes were contained in a DL ontology, the satisfiability test could have been actually performed. Hence, the separation of ontologies using $\mathcal{E}$-Connections naturally enhances the effect of caching for avoiding unnecessary subsumption tests.
- **Separation of non-absorbable GCIs**: When general inclusion axioms are present in a knowledge base, a disjunction must be added to the label of each node for each GCI. The resulting increase of non-determinism in the tableau expansion dramatically affects the performance of the DL reasoner. In order to overcome this problem, modern DL reasoners implement an optimization technique called *absorption* (28), which allows to transform many GCIs into primitive definition axioms. However, although absorption typically allows to eliminate most of the GCIs in a knowledge base, many general axioms may still remain. Non-absorbable GCIs, even in a reduced number, can notably degrade the performance of reasoners. In an $\mathcal{E}$-Connection, non-absorbable GCIs are typically also spread among the different ontologies of the combination. When performing the tableau expansion, a GCI *only* adds a disjunction to the nodes corresponding to the ontology the GCI belongs to, whereas, in the case of a single ontology, the same GCI would add a disjunction in *all* the nodes during the expansion.
- **Separation of ABox individuals**: Typical reasoning services when ABoxes are present are *instantiation* (checking if an individual is an instance of a concept), and *retrieval* (computing all the instances of a certain concept). The presence of a large ABoxes degrades significantly the performance of DL reasoners. When using $\mathcal{E}$-Connections, the ABox axioms and individuals are partitioned among the different ontologies in the combination. This separation is important since it may spare the application of a large number of rules in many satisfiability tests.
- **Optimization of blocking and better use of certain optimization techniques**: Blocking ensures the correct termination of tableau algorithms by limiting the tree expansion depth, which otherwise would become infi-

nite. However, when dealing with logics like $\mathcal{SHIQ}$, the blocking condition is quite sophisticated and blocking may occur late in the tableau expansion. Having a more permissive blocking condition is important to establish blocks at a shallower depth, and can substantially increase performance. Although the restrictions can be made less strict (29), the resulting blocking condition still remains less efficient than subset or equality blocking, which unfortunately are not sound for logics like $\mathcal{SHIQ}$. However, in an $\mathcal{E}$-Connection, blocking is optimized for each of the components, which implies, for example, that subset blocking is still applicable in a component without inverse properties, even if such a component is connected to a $\mathcal{SHIQ}$ component. On the other hand, certain optimization techniques are not valid for certain logics. For example, caching the satisfiability status of nodes within a tableau expansion cannot be performed in logics containing inverse roles. For the same reason as in blocking, optimizations like caching could still be applied to some of the component ontologies in the combination.

Finally, when decomposing a $\mathcal{SHOIN}(\mathcal{D})$ ontology into an $\mathcal{E}$-Connection, the resulting $\mathcal{E}$-Connection is likely to be written in one of the combined languages we have presented, for which we provide a tableau-based decision procedure. For example, let us consider again the Wine Ontology. The ontology is written in $\mathcal{SHION}(\mathcal{D})$, whereas its corresponding $\mathcal{E}$-Connected version is represented in the language $\mathcal{C}^{\mathcal{E}}_{\mathcal{HN}}(\mathcal{SIO}(\mathcal{D}))$, since the logic underlying the component ontologies is $\mathcal{ALR}_{+}\mathcal{IO}$ for regions, $\mathcal{AL}(\mathcal{D})$ for years and wineries, $\mathcal{ALCO}$ for wine descriptors, $\mathcal{ALO}$ for grapes and $\mathcal{ALC}$ for wines. Such a language is a fragment of $\mathcal{C}^{\mathcal{E}}_{\mathcal{HN}}(\mathcal{SHIN}(\mathcal{D}), \mathcal{SHON}(\mathcal{D}), \mathcal{SHIO}(\mathcal{D}))$, that we can handle algorithmically.

Note that the expressivity of the original ontology is split among the different component ontologies and the link properties. This fact can be exploited for optimizations.

7 Related Work

The most prominent formalism that has been proposed so far for combining and linking OWL ontologies is the Distributed Description Logics (DDL) (30) framework, which resulted in an extension of Web Ontology Language, called C-OWL (31).

Distributed Description Logics is a formalism for combining different DL knowledge bases in a loosely coupled information system. The idea of the combination is to preserve the "identity" and independence of each local ontology. The coupling is established by allowing a new set of inter-ontology axioms, called bridge rules. A bridge rule is an expression of one of the follow-

ing forms [16] :

$$C_i \xrightarrow{\sqsubseteq} C_j; \qquad ; C_i \xrightarrow{\sqsupseteq} C_j \; ; \qquad a_i \rightarrow b_j$$

Where, C_i, C_j are classes and a_i,b_j individuals in the ontologies O_i,O_j respectively.

From the modeling point of view, bridge rules have been conceived for establishing directional ("view dependent") subsumption relationships between classes and correspondences between individuals in different ontologies. The motivation for bridge rules, hence, covers a wide variety of intuitive modeling scenarios on the Semantic Web. For example, suppose that in ontology O_1 we have a class "Beer" defined as a subclass of "GermanProduct" and which does not exist in ontology O_2. However, in ontology O_2 there is a class "Drink", but there is no class modeling the concept of "Beer". Both ontologies could be linked by using a bridge rule stating that "Beer" in ontology O_1 is a subclass of "Drink" in O_2 .

These kind of modeling scenarios are appealing, because they nicely fit in the vision of the Semantic Web in which the notion of a universal upper ontology is abandoned by the notion of a web of distributed, linked and independently developed ontologies. Researchers in the Semantic Web community are starting to realize the importance of extending OWL with a suitable formalism that provides inter-ontology mappings with a precise logical semantics. A first attempt in this direction resulted in C-OWL (31), a syntax and semantic extension of OWL that adds the DDL formalism to the language.

However, C-OWL, as presented in (31) has some difficulties. First, no reasoning support is provided for the language. Extending the existing tableau reasoners to deal with such an extension is certainly a crucial issue. Second, some expressive features that are beyond DDLs (and also beyond $\mathcal{E}$-Connections), like the inclusion of inter-ontology role subsumption statements, are included in the language without the required discussion. This kind of expressive power could even make the reasoning undecidable.

Third, and most importantly, C-OWL fails to model certain crucial propertes of subsumption relations. For example, consider the ontology Figure 3a. The ontology O defines two disjoint concepts $Flying$ and $NonFlying$, the concept $Bird$ and the concept $Penguin$. The axioms in O state that all birds fly and also that a penguin is a bird, but it does not fly. In this case, in which all the axioms are gathered in a single logical space, the concept $Penguin$ would be clearly unsatisfiable [17] .

[16] We will consider bridge rules without complete individual correspondences along the paper.

[17] A Class is unsatisfiable in an ontology if it is evaluated to the empty set by every

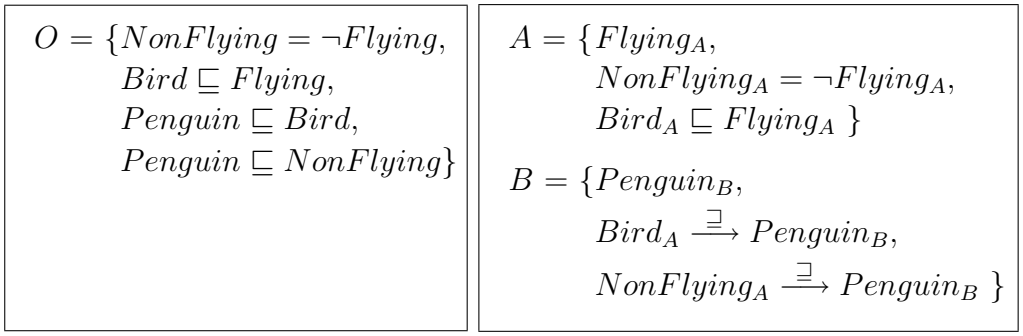

Fig. 3. Figure (a) shows a single ontology where concept *Penguin* is unsatisfiable. Figure (b) shows the same definition with bridge rules where contradiction cannot be detected anymore

Now, imagine that we split the knowledge about the domain in two coupled ontologies, shown in Figure 3b. The ontology A states that the concepts $Flying$ and $NonFlying$ are disjoint and states that all birds fly. On the other hand, the ontology B defines the concept $Penguin$ and states, using DDL subsumption links, that a penguin does not fly and that a penguin is a bird. However, it is easy to see by direct application of the semantics that in this case the obvious (and relevant) contradiction is not detected, and $Penguin$ is satisfiable in the coupled system.

The problem is that bridge rules can be reduced to simple axioms in an $\mathcal{E}$-Connection with a single link property, which is "hidden" in the syntax of bridge rules. A bridge rule always involves a restriction on that implicit link property. Intuitively, there is nothing contradictory in these bridge rules in the same way that there is no contradiction between two axioms like $Father \sqsubseteq \exists hasChild.Male$ and $Father \sqsubseteq \exists hasChild.\neg Male$ in an ordinary ontology.

This result, together with the fact that inter-ontology subsumption relations do not propagate transitively, shows that DDLs can be misleading and counter-intuitive, and do not seem to capture the notion of subsumption links across a "Web of ontologies". The example suggests that a formalism for dealing with inter-ontology subsumption relationships is still lacking. $\mathcal{E}$-Connections, though suitable for covering a wide variety of relevant modeling scenarios in the Semantic Web context, do not capture the idea of linking ontologies with subsumption relationships, but, as opposed to DDLs, were not conceived for such a purpose.

Finally, it is also worth mentioning CYC microtheories (32). CYC is partitioned into sets of axioms, called contexts or microtheories, so that each CYC entity must be contained in at least one of them. Whenever CYC is asked a question, or has to do some reasoning, the task is always performed in some

model of the ontology.

particular context. CYC microtheories are very similar to imports in OWL in that a given microtheory can be used by other microtheories, in which case all the axioms in the used (say, imported) microtheory are brought into the importing context. If there is an axiom somewhere in CYC that is not in that Microtheory, then CYC will not use that axiom: it is inaccessible.

8 Conclusion and Future Work

In this paper, we have presented $\mathcal{E}$-Connections as a suitable formalism for combining OWL ontologies. We have discussed the applicability and usefulness of $\mathcal{E}$-Connections as a combination technique for many application scenarios.

In order to integrate $\mathcal{E}$-Connections in the Semantic Web, we have extended the syntax and semantics of the OWL-DL recommendation and we have discussed the issues of handling URIs and imports in this new framework.

We have provided suitable tool support for browsing and editing $\mathcal{E}$-Connected ontologies in SWOOP.

We have shown how to reason with certain families of $\mathcal{E}$-Connections and proved that it is possible to implement our tableau algorithms for $\mathcal{E}$-Connections as an extension of existing DL reasoners, as shown by our implementation in the Pellet system. Our initial empirical results suggest that reasoning over expressive $\mathcal{E}$-Connections is reasonably efficient and, in practice, it is not harder than reasoning with OWL itself.

Finally, we have also identified the limitations of $\mathcal{E}$-Connections as a combination technique, in particular for tasks, such as ontology refinement, that involve connecting ontologies dealing with highly overlapping domains.

The results presented in this paper raise a number of issues for future work. First, our implementation and initial results show that $\mathcal{E}$-Connections suggest new optimization techniques. The empirical evaluation of these optimizations as well as the development and implementation of new ones is a priority in our future research agenda.

Second, we are investigating the applications of $\mathcal{E}$-Connections for *partitioning* automatically OWL ontologies into its relevant sub-parts. The theoretical and empirical results (33) we have obtained so far are very encouraging.

Third, we aim to explore the design and implementation of practical algorithms for combinations of Description Logics with spatial and temporal logics that are within the ADS framework. These combinations are important for many applications, since OWL is not a suitable formalism for representing

temporal and spatial information. For instance, in the Wine Ontology example, the ontology about regions could be represented using a qualitative spatial logic, like $\mathbf{S4_u}$, instead of using OWL. In (8) the decidability of such combinations was proved and the development of practical algorithms will provide a strong motivation for bringing these formalisms to the Semantic Web.

Finally, although rules formalisms are beyond the ADS framework, we strongly believe that a generalization of many (decidable) rules languages is possible and that there will be decidable combinations of such a generalization with ADSs in the spirit of $\mathcal{E}$-Connections. Such a result would open new horizons in the integration of ontologies and rules on the Semantic Web.

References

[1] F. Baader, I. Horrocks, U. Sattler, Description logics as ontology languages for the semantic web, in: D. Hutter, W. Stephan (Eds.), Festschrift in honor of Jörg Siekmann, Lecture Notes in Artificial Intelligence, Springer-Verlag, 2003.

[2] I. Horrocks, P. F. Patel-Schneider, F. van Harmelen, From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language, J. of Web Semantics 1 (1) (2003) 7–26.

[3] P. Patel-Schneider, P. Hayes, I.Horrocks, Web ontology language OWL Abstract Syntax and Semantics, W3C Recommendation (2004).

[4] M. Smith, C. Welty, D. McGuiness, OWL Web Ontology Language Guide, W3C Recommendation (2004).

[5] B. Cuenca, B. Parsia, E. Sirin, Working with multiple ontologies on the semantic web, in: Proc. of the 3thrd International Semantic Web Conference (ISWC 2004), Vol. 3298 of Lecture Notes In Computer Science, Springer Verlag, 2004.

[6] H. StuckenSchmidt, M. Klein, Structure-based partitioning of large class hierarchies, in: Proceedings of the Third International Semantic Web Conference (ISWC-2004), Springer Verlag, 2004.

[7] H. StuckenSchmidt, M. Klein, Integrity and change in modular ontologies, in: Proceedings of the Eighteenth Joint Conference on Artificial Intelligence (IJCAI-2003), Morgan Kaufmann, 2003.

[8] O. Kutz, C. Lutz, F. Wolter, M. Zakharyaschev, $\mathcal{E}$-Connections of Abstract Description Systems, Artificial Intelligence 156 (2004) 1–73.

[9] O. Kutz, $\mathcal{E}$-Connections and logics of distance, Ph.D. thesis, University of Liverpool (2004).

[10] F. Baader, C. Lutz, H.Sturm, F.Wolter, Fusions of description logics and abstract description systems, Journal of Artificial Intelligence Research (JAIR) 16 (2003) 1–58.

[11] A. Kalyanpur, B. Parsia, J. Hendler, A tool for working with web ontolo-

gies, International Journal on Semantic Web and Information Systems 1 (1).

[12] E. Sirin, B.Parsia, Pellet: An OWL-DL reasoner, http://www.mindswap.org/2003/pellet (2004).

[13] J. Golbeck, G. Fragoso, F. Hartel, J. Hendler, B. Parsia, J. Oberthaler, The national cancer institute's thesaurus and ontology, Journal of Web Semantics 1, (2003) (1).

[14] S. Bechhofer, P. Lord, R.Volz, Cooking the semantic web with the OWL API, in: Proc. of the Second International Semantic Web Conference (ISWC-2003), 2003.

[15] D. Brickley, R. Guha, Resource description framework (RDF) model and syntax specification, W3C Recommendation (2004).

[16] B. Cuenca-Grau, B. Parsia, E.Sirin, Tableau algorithms for e-connections of description logics, Tech. rep., UMIACS, available at http://www.mindswap.org/2004/multipleOnt/papers/EconnTableau.ps (2004).

[17] I. Horrocks, U. Sattler, S. Tobies, Practical reasoning for very expressive description logics, Logic Journal of the IGPL 8 (3) (2000) 239–263.

[18] I. Horrocks, Using an expressive description logic: FaCT or fiction?, in: Proc. of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR-98), Morgan Kaufman, 1998, pp. 636–647.

[19] V.Haarslev, R.Moeller, Racer system description, in: Proc. of the Joint Conf. on Automated Reasoning (IJCAR 2001). Volume 2083 of Lecture Notes in Artificial Intelligence, pages 701-705, 2001.

[20] I. Horrocks, Optimising tableaux decision procedures for description logics, Ph.D. thesis, University of Manchester (1997).
URL `download/1997/phd-2sss.ps.gz`

[21] I. Horrocks, U. Sattler, A description logic with transitive and inverse roles and role hierarchies, Journal of Logic and Computation 9 (3) (1999) 385–410.

[22] I. Horrocks, U. Sattler, Ontology reasoning in the $\mathcal{SHOQ}(\mathcal{D})$ description logic, in: B. Nebel (Ed.), Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001), Morgan Kaufmann, 2001, pp. 199–204.

[23] J. Hladik, J. Model, Tableau systems for $\mathcal{SHIO}$ and $\mathcal{SHIQ}$, in: V. Haarslev, R. Möller (Eds.), Proceedings of the 2004 International Workshop on Description Logics (DL 2004), CEUR, 2004, available from `ceur-ws.org`.

[24] I. Horrocks, U. Sattler, A tableaux decision procedure for SHOIQ, in: Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005), 2005.

[25] F. Baader, U. Sattler, An overview of tableau algorithms for description logics, Studia Logica 69 (2001) 5–40.

[26] F. Baader, W. Nutt, Basic description logics, in: F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. F. Patel-Schneider (Eds.), The Description

Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, 2003, pp. 43–95.

[27] F. Baader, J. Hladik, C. Lutz, F. Wolter, From tableaux to automata for description logics, Fundamenta Informaticae 57 (2003) 1–33.

[28] I. Horrocks, Implementation and optimisation techniques, in: F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, 2003, pp. 306–346.

[29] I. Horrocks, U. Sattler, Optimised reasoning for $\mathcal{SHIQ}$, in: Proc. of the 15th European Conference on Artificial Intelligence (ECAI-2002), 2002.

[30] A. Borgida, L. Serafini, Distributed description logics: Assimilating information from peer sources, Journal of Data Semantics, 1:153-184 1 (2003) 153–184.

[31] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, H. Stuckenschmidt, C-OWL: Contextualizing ontologies, in: Proc. of the Second International Semantic Web Conference (ISWC 2003), 2003.

[32] OpenCYC, Contexts in CYC, http://www.cyc.com/cycdoc/course/what-is-a-context.html.

[33] B. Cuenca-Grau, B. Parsia, E.Sirin, A.Kalyanpur, Automatic partitioning of owl ontologies using e-connections, Tech. rep., UMIACS, available at http://www.mindswap.org/2004/multipleOnt/papers/Partition.pdf (2005).

[34] P. F. Patel-Schneider, I. Horrocks, DLP and FaCT, in: N. V. Murray (Ed.), Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'99, no. 1617 in Lecture Notes in Artificial Intelligence, Springer-Verlag, 1999, pp. 19–23.

[35] D. Calvanese, G. De Giacomo, Expressive description logics, in: F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation and Applications, Cambridge University Press, 2003, pp. 178–218.

[36] T. Berners-Lee, J. Hendler, O. Lassila, The semantic web, Scientific American 284 (2001) 34–43.

[37] S. Tobies, Complexity results and practical algorithms for logics in knowledge representation, Ph.D. thesis, RWTH Aachen (2001).

[38] U. Sattler, A concept language extended with different kinds of transitive roles, in: B. Nebel (Ed.), Proc. of the 20th German Annual Conf. on Artificial Intelligence (KI 2001), Vol. 1137 of Lecture Notes In Artificial Intelligence, Springer Verlag, 2001, pp. 199–204.