

LABORATOIRE



INFORMATIQUE, SIGNAUX ET SYSTÈMES
DE SOPHIA ANTIPOLIS
UMR 6070

EXTENDING OWL-DL WITH IDENTIFICATION CONSTRAINTS

Thi Dieu Thu NGUYEN, Nhan LE-THANH

Projet MAINLINE

Rapport de recherche
ISRN I3S/RR-2007-03-FR

Janvier 2007

RÉSUMÉ :

Les contraintes d'identification ont toujours une haute position dans la liste des besoins des utilisateurs dans l'environnement de Web sémantique. En dépit de beaucoup recherche en théorie, ces contraintes ne sont pas encore disponibles dans des langages d'ontologies Web. Pour répondre à ce besoin, nous présentons un langage d'extension décidable du langage OWL-DL, une sorte de OWL (Ontology Web Language) - un langage populaire pour concevoir les ontologies dans l'environnement de Web sémantique, proposé par le consortium World Wide Web (W3C), à savoir OWL-K, qui fournit de telles contraintes. La possibilité de décision d'OWL-K est également présentée par l'intermédiaire d'un nouvel langage SHOINK(D) qui est une extension de SHOIN(D), le langage de logique de description au-dessous de OWL-DL, de telle sorte que la représentation des contraintes d'identification soit cohérente.

MOTS CLÉS :

Langages d'ontologies Web, Web Sémantique, Logiques de Description, Contraintes d'Identification

ABSTRACT:

Identification constraints have always been high on the list of requests from users in the Semantic Web environment. Despite many researches in theory, this kind of constraints, however, has not been available in Web ontology languages yet. To meet this requirement, this report introduces a decidable extension of OWL-DL, a kind of OWL (Ontology Web Language) - a popular language for designing the ontologies in the Semantic Web environment recommended by World Wide Web Consortium (W3C), namely OWL-K, that supports such constraints. The decidability of OWL-K is also presented via a new description logic language SHOINK(D) which is an extension of SHOIN(D), the description logic language underlying OWL-DL, so that the representation of identification constraints is meaningful.

KEY WORDS :

Web ontology languages, Semantic Web, Description Logics, Identification constraints

Extending OWL-DL with Identification Constraints

Thi Dieu Thu NGUYEN — Nhan LE-THANH

*Projet MAINLINE - Laboratoire I3S (CNRS - UNSA)
Les Algorithmes - Bât Euclide B
2000 route des Lucioles – B.P. 121
F-06903 Sophia Antipolis Cedex
tnguyen@i3s.unice.fr;nhan.-le-thanh@unice.fr*

RÉSUMÉ. Les contraintes d'identification ont toujours une haute position dans la liste des besoins des utilisateurs dans l'environnement de Web sémantique. En dépit de beaucoup recherche en théorie, ces contraintes ne sont pas encore disponibles dans des langages d'ontologies Web. Pour répondre à ce besoin, nous présentons un langage d'extension décidable du langage OWL-DL¹, à savoir OWL-K, qui fournit de telles contraintes. La possibilité de décision d'OWL-K est également présentée par l'intermédiaire d'un nouvel langage $\mathcal{SHOIN}K(\mathbf{D})$ qui est une extension de $\mathcal{SHOIN}(\mathbf{D})$, le langage de logique de description au-dessous de OWL-DL, de telle sorte que la représentation des contraintes d'identification soit cohérente.

ABSTRACT. Identification constraints have always been high on the list of requests from users in the Semantic Web environment. Despite many researches in theory, this kind of constraints, however, has not been available in Web ontology languages yet. To meet this requirement, this report introduces a decidable extension of OWL-DL², namely OWL-K, that supports such constraints. The decidability of OWL-K is also presented via a new description logic language $\mathcal{SHOIN}K(\mathbf{D})$ which is an extension of $\mathcal{SHOIN}(\mathbf{D})$, the description logic language underlying OWL-DL, so that the representation of identification constraints is meaningful.

MOTS-CLÉS : Langages d'ontologies Web, Web Sémantique, Logiques de Description, Contraintes d'Identification.

1. OWL-DL est une sorte de OWL (Ontology Web Language), un langage populaire pour concevoir les ontologies dans l'environnement de Web sémantique, proposé par le consortium World Wide Web (W3C).

2. OWL-DL is a kind of OWL (Ontology Web Language), a popular language for designing the ontologies in the Semantic Web environment, recommended by World Wide Web Consortium (W3C).

2 Rapport de recherche – 2006/2

KEYWORDS: Web ontology languages, Semantic Web, Description Logics, Identification constraints.

Table des matières

Liste des tableaux

1. Introduction

Identification constraints (ICs) are very important in applications and so in the Semantic Web ontologies because all the things in the real world, in some way, need to be identified. Since long time, identification constraints have been modeled in Database (DB) schemas under the name *key*. However, up to now this important feature has not been fully captured in the Semantic Web yet.

Particularly, the Web Ontology language recommended by W3C¹ for expressing ontologies in the Semantic Web is OWL (?). Considered as its most important sub-language, OWL-DL (OWL-Description Logic) is designed with the purpose of enough expressivity to deal with the problem of heterogeneity of sources and of good formalization to be able to be used in automated reasoning. However, the capacity of expressing ICs of OWL-DL is seriously limited. Actually, OWL-DL only provides a “spare” mechanism to express the functional dependency between two elements. It cannot describe 1-1 relations between two elements (so called *simple keys* in DB schemas) or between a set of elements and an element (so called *compound keys* in DB schemas).

As the underlying of OWL-DL is a description logic (DL) language, this limitation has also pointed out a gap in the expressive power of DLs. There have been many efforts to add these constraints to DLs (?, ?, ?, ?, ?). However, to the best of our knowledge, there exist no DLs with practical reasoning procedures that support ICs.

Accordingly, the need of IC support in the Semantic Web has always been high on the list of requests from the Semantic Web community (?).

To meet this requirement, this paper introduces a decidable extension of OWL-DL, namely OWL-K (K stands for “key”), that supports such constraints. Decidability of OWL-K is also presented via a new DL language $\mathcal{SHOINK}(\mathbf{D})$ which is an extension of $\mathcal{SHOIN}(\mathbf{D})$, the DL language underlying OWL-DL.

This paper makes three main contributions. Firstly, it provides a review of OWL-DL with its capability of IC representation. From that the requirements necessary for OWL to support ICs are identified. Secondly, and most importantly, it presents a decidable extension of OWL-DL, called OWL-K, to support ICs. Thirdly, the underpinning DL of OWL-K, $\mathcal{SHOINK}(\mathbf{D})$, is presented.

The rest of this paper is organized as follows. Section 2 briefly introduces the OWL-DL language. Section 3 describes the limitations of OWL-DL in representing

1. World Wide Web Consortium, <http://www.w3.org/>

Tableau 1. *OWL-DL class, property descriptions*

Abstract syntax	DL Syntax	Semantics
Class(A) Class(owl :Thing) Class(owl :Nothing)	A ⊤ ⊥	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ $\text{owl :Thing}^{\mathcal{I}} = \Delta^{\mathcal{I}}$ $\text{owl :Nothing}^{\mathcal{I}} = \emptyset$
intersectionOf($C_1 \dots C_n$) unionOf($C_1 \dots C_n$) complementOf(C) oneOf($o_1 \dots o_n$)	$C_1 \sqcap \dots \sqcap C_n$ $C_1 \sqcup \dots \sqcup C_n$ $\neg C$ $\{o_1\} \sqcup \dots \sqcup \{o_n\}$	$(C_1 \sqcap \dots \sqcap C_n)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap \dots \cap C_n^{\mathcal{I}}$ $(C_1 \sqcup \dots \sqcup C_n)^{\mathcal{I}} = C_1^{\mathcal{I}} \cup \dots \cup C_n^{\mathcal{I}}$ $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$ $(\{o_1\} \sqcup \dots \sqcup \{o_n\})^{\mathcal{I}} = \{o_1^{\mathcal{I}}, \dots, o_n^{\mathcal{I}}\}$
restriction(R someValuesFrom(C)) restriction(R allValuesFrom(C)) restriction(R hasValue(o)) restriction(R minCardinality(n)) restriction(R maxCardinality(n))	$\exists R.C$ $\forall R.C$ $\exists R.\{o\}$ $\geq nR$ $\leq nR$	$(\exists R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y. \langle x, y \rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$ $(\forall R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y. \langle x, y \rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$ $(\exists R.\{o\})^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \langle x, \{o\}^{\mathcal{I}} \rangle \in R^{\mathcal{I}}\}$ $(\geq nR)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#\{y. \langle x, y \rangle \in R^{\mathcal{I}}\} \geq n\}$ $(\leq nR)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#\{y. \langle x, y \rangle \in R^{\mathcal{I}}\} \leq n\}$
restriction(U someValuesFrom(d)) restriction(U allValuesFrom(d)) restriction(U hasValue(v)) restriction(U minCardinality(n)) restriction(U maxCardinality(n))	$\exists U.d$ $\forall U.d$ $\exists U.v$ $\geq nU$ $\leq nU$	$(\exists U.d)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y. \langle x, y \rangle \in U^{\mathcal{I}} \wedge y \in d^{\mathcal{D}}\}$ $(\forall U.d)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y. \langle x, y \rangle \in U^{\mathcal{I}} \rightarrow y \in d^{\mathcal{D}}\}$ $(\exists U.v)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \langle x, v \rangle \in U^{\mathcal{I}}\}$ $(\geq nU)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#\{y. \langle x, y \rangle \in U^{\mathcal{I}}\} \geq n\}$ $(\leq nU)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#\{y. \langle x, y \rangle \in U^{\mathcal{I}}\} \leq n\}$
ObjectProperty(S) inverseOf(S) DatatypeProperty(U)	S S^- U	$S^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ $(S^-)^{\mathcal{I}} = \{\langle x, y \rangle \mid \langle y, x \rangle \in S^{\mathcal{I}}\}$ $U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta_{\mathcal{D}}$

ICs and shows the requirements that the extended language must satisfy to express ICs. Section 4 presents the design of OWL-K language. Section 5 describes the decidability of the extended language by showing the DL language underlying OWL-K, *SHOINK(D)*. Section 6 concludes the paper and suggests some future work.

2. Overview of OWL-DL Language

OWL-DL, a species of OWL (?), is designed to support the Description Logic business segment and to provide a language subset that has desirable computational properties for reasoning systems. It has the same set of constructors as OWL, but restricts them to be used in a way satisfying decidable inference. What makes OWL-DL a Semantic Web language, however, is not its semantics, which are quite standard for a DL, but its RDF/XML exchange syntax besides an abstract frame-like syntax (see section 4 of (?)).

The underpinning of OWL-DL is *SHOIN(D)* DL, which is an extension of *SHOQ(D)* (?) with inverse roles and restricted to unqualified number restrictions. Abstract syntax, DL syntax and Semantics of OWL-DL descriptions and axioms can be seen in table ?? and table ??, where A is a class URI reference; C, C_1, \dots, C_n are class descriptions; S is an object property URI reference; R, R_1, \dots, R_n are object property descriptions; o, o_1, \dots, o_n are individual URI references; d is a data range; U is a datatype property; # denotes cardinality; \mathcal{I} is the interpretation function; $\Delta^{\mathcal{I}}$ is the individual domain and $\Delta_{\mathcal{D}}$ is the domain of data values. More details of OWL-DL language and its semantics can be found in (?).

As shown in table ??, OWL-DL can declare classes. OWL classes can be specified as logical combinations (intersections, unions, or complements) of other classes, as

Tableau 2. OWL-DL axioms

Abstract syntax	DL Syntax	Semantics
Class(A partial $C_1 \dots C_n$) Class(A complete $C_1 \dots C_n$) EnumeratedClass(A $o_1 \dots o_n$) SubClassOf(C_1, C_2) EquivalentClasses($C_1 \dots C_n$) DisjointClasses($C_1 \dots C_n$)	$A \sqsubseteq C_1 \sqcap \dots \sqcap C_n$ $A \equiv C_1 \sqcap \dots \sqcap C_n$ $A \equiv \{o_1\} \sqcap \dots \sqcap \{o_n\}$ $C_1 \sqsubseteq C_2$ $C_1 \equiv \dots \equiv C_n$ $C_i \sqsubseteq \neg C_j,$ $(1 \leq i < j \leq n)$	$A^{\mathcal{I}} \subseteq C_1^{\mathcal{I}} \sqcap \dots \sqcap C_n^{\mathcal{I}}$ $A^{\mathcal{I}} \equiv C_1^{\mathcal{I}} \sqcap \dots \sqcap C_n^{\mathcal{I}}$ $A^{\mathcal{I}} \equiv \{o_1^{\mathcal{I}}, \dots, o_n^{\mathcal{I}}\}$ $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ $C_1^{\mathcal{I}} = \dots = C_n^{\mathcal{I}}$ $C_i^{\mathcal{I}} \cap C_j^{\mathcal{I}} = \emptyset,$ $(1 \leq i < j \leq n)$
DatatypeProperty(U super(U_1) ... super(U_n)) domain(C_1) ... domain(C_m) range(d_1) ... range(d_k) [Functional] SubPropertyOf(U_1, U_2) EquivalentProperties($U_1 \dots U_n$) ObjectProperty(R super(R_1) ... super(R_n)) domain(C_1) ... domain(C_m) range(C_1) ... range(C_k) [Symmetric] [Functional] [InverseFunctional] [Transitive] SubPropertyOf(R_1, R_2) EquivalentProperties($R_1 \dots R_n$) AnnotationProperty(R)	$U \sqsubseteq U_i, (1 \leq i \leq n)$ $\geq 1U \sqsubseteq C_i, (1 \leq i \leq m)$ $\top \sqsubseteq \forall U.d_i, (1 \leq i \leq k)$ $\top \sqsubseteq \leq 1U$ $U_1 \sqsubseteq U_2$ $U_1 \equiv \dots \equiv U_n$ $R \sqsubseteq R_i, (1 \leq i \leq n)$ $\geq 1R \sqsubseteq C_i, (1 \leq i \leq m)$ $\top \sqsubseteq \forall R.C_i, (1 \leq i \leq k)$ $R \equiv R^-$ $\top \sqsubseteq \leq 1R$ $\top \sqsubseteq \leq 1R^-$ $\text{Trans}(R)$ $R_1 \sqsubseteq R_2$ $R_1 \equiv \dots \equiv R_n$	$U^{\mathcal{I}} \subseteq U_i^{\mathcal{I}}, (1 \leq i \leq n)$ $U^{\mathcal{I}} \subseteq C_i^{\mathcal{I}} \times \Delta_{\mathbf{D}}^{\mathcal{I}}, (1 \leq i \leq m)$ $U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times d_i^{\mathcal{I}}, (1 \leq i \leq k)$ $\#\{ \langle x, y \rangle \mid \{y.\langle x, y \rangle \in U^{\mathcal{I}} \} \leq 1 \forall x \in \Delta^{\mathcal{I}} \}$ $U_1^{\mathcal{I}} \subseteq U_2^{\mathcal{I}}$ $U_1^{\mathcal{I}} = \dots = U_n^{\mathcal{I}}$ $R^{\mathcal{I}} \subseteq R_i^{\mathcal{I}}, (1 \leq i \leq n)$ $R^{\mathcal{I}} \subseteq C_i^{\mathcal{I}} \times \Delta^{\mathcal{I}}, (1 \leq i \leq m)$ $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times C_i^{\mathcal{I}}, (1 \leq i \leq k)$ $R^{\mathcal{I}} = (R^-)^{\mathcal{I}}$ $\#\{ \langle x, y \rangle \mid \{y.\langle x, y \rangle \in R^{\mathcal{I}} \} \leq 1 \forall x \in \Delta^{\mathcal{I}} \}$ $\#\{ \langle x, y \rangle \mid \{y.\langle x, y \rangle \in (R^-)^{\mathcal{I}} \} \leq 1 \forall x \in \Delta^{\mathcal{I}} \}$ $R^{\mathcal{I}} = (R^{\mathcal{I}})^+$ $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$ $R_1^{\mathcal{I}} = \dots = R_n^{\mathcal{I}}$
Individual(o) type(C_1) ... type(C_n) value(R_1 o_1) ... value(R_n o_n) SameIndividual($o_1 \dots o_n$) DifferentIndividuals($o_1 \dots o_n$)	$o \in C_i$ $\langle o, o_i \rangle \in R_i$ $o_1 = \dots = o_n$ $o_i \neq o_j$	$o^{\mathcal{I}} \in C_i^{\mathcal{I}}$ $\langle o^{\mathcal{I}}, o_i^{\mathcal{I}} \rangle \in R_i^{\mathcal{I}}$ $o_1^{\mathcal{I}} = \dots = o_n^{\mathcal{I}}$ $o_i^{\mathcal{I}} \neq o_j^{\mathcal{I}}$

enumerations of specified objects or as restrictions on a particular property so that all the values for the property in instances of the class must belong to a certain class (or datatype); at least one value must come from a certain class (or datatype); there must be at least certain specific values; there must be at least or at most a certain number of distinct values. Table ?? shows that classes can be organized in a subsumption (sub-class) hierarchy. OWL-DL can also declare properties with their domains and ranges, organize them into a subproperty hierarchy. It can also state whether a property is transitive, symmetric, functional, or inverse of another property. An OWL-DL ontology can be seen as a DL knowledge base (KB), which consists of a set of axioms, including class axioms, property axioms and individual axioms (also called ‘‘facts’’).

3. OWL-DL with Identification Constraints

In this section, we will review the capability of OWL-DL in describing Identification constraints, show its limitations and identify the requirements that an extension of OWL-DL must satisfy.

In the sense of OWL-DL, ICs can be defined as to state that a certain set of properties uniquely identifies the instances of a given class. Essentially, these constraints put a 1-1 relation between sets of values of respective properties and instances of a

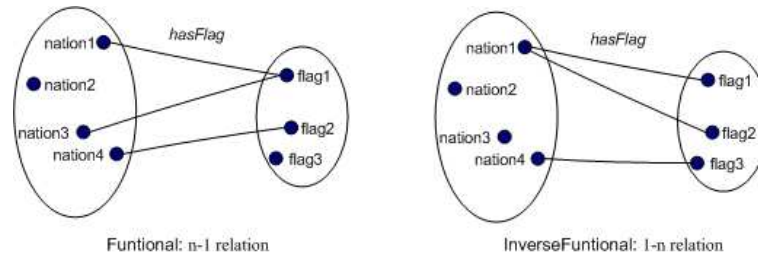


Figure 1. Examples showing the limitations of OWL-DL in representing ICs

class. Although supporting considerable expressive power to the Semantic Web, the mechanism to express this kind of constraints in OWL-DL is still very seriously limited. In particular, OWL-DL provides two constructors, namely (in abstract syntax) `Functional` and `InverseFunctional`, which can be used to link individuals together.

If a property, P , is tagged as `Functional` then for all x , y , and z : $P(x,y)$ and $P(x,z)$ implies $y = z$. For example, if the property `hasFlag` is characterized as `Functional`, then each nation has at most one flag ; if a nation has “2 flags”, they must be the same (see the illustration of `Functional` constructor in figure ??).

If a property, P , is tagged as `InverseFunctional` then for all x , y and z : $P(y,x)$ and $P(z,x)$ implies $y = z$. For example, if the property `hasFlag` above is characterized as `InverseFunctional`, then two nations could be inferred to be identical based on having the same flag (see the illustration of `InverseFunctional` in figure ??).

Suppose that one would like to identify uniquely each nation by its flag. The constructor `Functional` shown above obviously does not respond to this requirement while `InverseFunctional` seems to fit well. Accordingly, one can think of `InverseFunctional` as defining ICs.

However, figure ?? shows that `InverseFunctional` does not require that two separate flags must correspond to separate nations. Consequently, one nation can be identified by different flags, which is not expected. Actually, the relation put by `InverseFunctional` is a 1-n relation whereas `Functional` represents n-1 relations. Therefore one can think of using both of these constructors to represent the notion of ICs. Applying both these constructors on the property `hasFlag`, one nation cannot be identified by different flags anymore.

Nevertheless, the relations put by `Functional` and `InverseFunctional` are not compulsory to all elements of the domain. As shown in figure ??, not all nations must have flags and vice versa. In the sense of DB schemas, we can say that these constructors allow for expressing null keys, which is never permitted in ICs. In our opinion, `InverseFunctional` and `Functional` can be stated more precisely as defining functional dependencies, but not ICs (which are stronger constraints).

Furthermore, the DL syntax and semantics of `InverseFunctional` and `Functional` show that these constructors are special kinds of number restrictions (see table ?? and ??). They essentially apply the restrictions to all elements of the universe, i.e., `owl:Thing` - so called *global* restrictions. That is why one cannot use these constructors to put dependencies on certain classes. For example, suppose that the classes `Nation` and `NationHistory` both share the property `hasFlag`. One would like to assert that `hasFlag` is a unique identifier for instances of `Nation`, but not for instances of `NationHistory`. `InverseFunctional` cannot express this.

Another limitation of OWL-DL is that `InverseFunctional` can only be set for `ObjectProperty` (properties for which values are individuals), but not for `DatatypeProperty` (properties for which values are data literal) (see table ??). That is, it can only put relations between classes, but not between a class and a datatype. Consequently, it is impossible in OWL-DL to assert that `hasCitycode` (which has the range of datatype `integer`) is an identifier for instances of class `City`.

In addition, these constructors are designed to put only relations between two elements (only possibly referring to *simple keys* in DB schemas), while those between an element and a set of elements (corresponding to *compound keys* in DB schemas) cannot be expressed. Let us revisit the class `NationHistory` mentioned above in example ??.

Example1 : One would like to state that instances of class `NationHistory` are uniquely identified by a couple of properties (`hasFlag`, `onDate`), where `hasFlag` is an `ObjectProperty` whose values are flags and `onDate` is a `DatatypeProperty` whose values are dates. The constraint described in this example is obviously out of the expressivity of OWL-DL.

As a result, to express ICs in the Semantic Web, a new mechanism must be designed that needs to satisfy the following requirements :

- It should provide the “real” ICs, i.e., 1-1 relations, but not functional dependencies ;
- It should put relations obligatory to all elements of the domain in question ;
- It should support ICs on specific classes, or so called non-global ICs ;
- It should provide ICs for both datatype properties and object properties ;
- It should support both kinds of constraints corresponding to simple and compound keys in DB schemas ;
- It should be detachable. So that the extension of OWL-DL does not influence the existing syntax of OWL-DL. The users of OWL are therefore able to switch off this utility if they do not want to use it ;
- The extension with this mechanism should be a *decidable* extension of OWL-DL.

The next section will introduce a such mechanism which makes an extended language of OWL-DL called OWL-K.

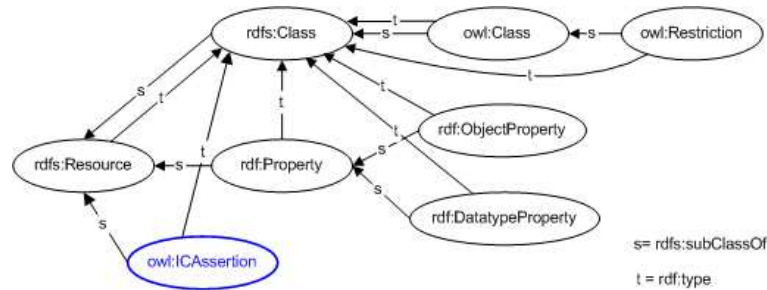


Figure 2. Class Hierarchy for OWL-K

4. Modeling ICs in OWL-K

In this section, we introduce a mechanism to represent ICs into OWL-DL, which creates an extension language called OWL-K. The new mechanism is designed separately from the existing syntax of OWL-DL so that it can be switched off if the users do not want to use it.

4.1. Vocabulary

In order to satisfy the requirements shown in section ??, we extend OWL-DL with IC assertions resulting in the language OWL-K. In this language, IC assertions are modeled as entities. They are neither classes (or concepts) nor properties (or roles). Hence IC assertions are defined as instances of a new class `owl:ICAssertion`, which is a subclass of `rdfs:Resource` (see figure ??).

Figure ?? shows information about the class hierarchy using a “nodes and arcs” graph representation of the RDF data model. If a class is a subset of another, then there is an `rdfs:subClassOf` (s) arc from the node representing the former class to the node representing the latter. Similarly, if a resource is an instance of a class, then there is an `rdf:type` (t) arc from the resource to the node representing the class (note that not all resources, classes and arcs are shown. We only show the principle resources, classes and arcs relating to our discussion for the extension. The rest is the same as for OWL-DL (?)).

As specified in the requirements (see section ??), an IC must put a constraint on a set of properties. Let’s see how OWL-DL provides property restrictions. Firstly, property restrictions are defined as classes while ICs do not create new classes. Secondly, OWL-DL distinguishes two kinds of property restrictions : value constraints and cardinality constraints. *Value constraints* put constraints on the range of a property when applied to this particular class description. *Cardinality constraints* put constraints on the number of values a property can take, in the context of this particular class descrip-

tion. So restriction constructs in OWL-DL put restrictions on only *one* property while an IC puts a restriction on a collection of properties. Restriction constructs in OWL-DL, therefore, do not agree with ICs. To express IC assertions, we introduce new kinds of restriction constructs, namely `owl :onClass` and `owl :byProperty`. `owl :onClass` is used to specify the class an IC is applied on. `owl :byProperty` is used to specify the property in the collection of properties identifying instances of a given class. Since ICs can be applied both to datatype properties and object properties, `owl :byProperty` is designed to have range both of the data-valued and object properties. Table ?? shows the vocabulary extension of OWL-K compared with OWL-DL.

Tableau 3. Vocabulary extension of OWL-K

Property name	Type	rdfs :domain	rdfs :range
<code>owl :onClass</code>	<code>rdf :ObjectProperty</code>	<code>owl :ICAssertion</code>	<code>owl :Class</code>
<code>owl :byProperty</code>	<code>rdf :Property</code>	<code>owl :ICAssertion</code>	<code>owl :Property</code>

4.2. Abstract Syntax

The abstract syntax is used to facilitate access to and evaluation of the language. It is specified by means of a version of Extended BNF (Backus Naur Form), which is defined in section 2 of (?).

IC assertions in OWL-K ontologies must be identifiable and referable like any other entity of ontologies. Hence, as for classes, properties and instances in OWL-DL ontologies, we associate with each IC assertion in an OWL-K ontology an identifier, which is a URI reference.

As the axioms modeled in OWL-DL, a new kind of axioms to express IC assertions is added as follows.

```

ICAssertionID ::= URIreference
axiom ::= 'ICAssertion(' ICAssertionID
         description
         propertyID {propertyID}'
propertyID ::= datavaluedPropertyID |
             individualvaluedPropertyID
    
```

With this syntax, the IC in example ?? can be described in the abstract syntax as follows :

Tableau 4. Mapping from OWL-K abstract syntax to DL syntax and semantics

Abstract syntax	DL syntax	Semantics
<code>ICAssertion(ICAssertionID C R₁ ... R_n)</code>	$(R_1, \dots, R_n \text{ Idfor } C)$	$\mathcal{I} \models (R_1, \dots, R_n \text{ Idfor } C)$ iff $\forall s, s' \in C^{\mathcal{I}}, \langle s, t_i \rangle, \langle s', t'_i \rangle \in R_i^{\mathcal{I}}$ and $t_i = t'_i \forall 1 \leq i \leq n$ then $s = s'$

Tableau 5. Transformation of IC assertion to triples

Abstract syntax S	Transformation - T(S)
ICAssertion(ICAssertionID description <i>propertyID</i> ₁ ... <i>propertyID</i> _n)	ICAssertionID rdf:type owl:ICAssertion. ICAssertionID rdf:type rdfs:Resource.[opt] ICAssertionID owl:onClass T(description). ICAssertionID owl:byProperty T(<i>propertyID</i> ₁). ... ICAssertionID owl:byProperty T(<i>propertyID</i> _n).

ICAssertion(NationHistoryIC NationHistory hasFlag onDate)

4.3. Semantics

The semantics of the abstract syntax above is defined by definition ???. Actually, this definition is provided by a constructor called **Idfor**, which is integrated in $\mathcal{SHOIN}(\mathbf{D})$, the underpinning DL of OWL-DL, and produces the underpinning of OWL-K, the DL $\mathcal{SHOINK}(\mathbf{D})$ (?). Note that in DLs we talk about *concepts*, *abstract roles* and *concrete roles* while in Web ontology languages we usually call them *classes*, *object properties* and *datatype properties* respectively.

Definition 4.1 (Identification constraint) *An identification constraint is a definition defined as in the following formula. The definition using the new constructor called **Idfor** is described like :*

$$(R_1, \dots, R_n \mathbf{Idfor} C)$$

where C is a concept, each R_i is a simple role (abstract or concrete) $\forall 1 \leq i \leq n$. The semantics of this definition are formally defined by the following interpretation : An interpretation \mathcal{I} satisfies the definition $(R_1, \dots, R_n \mathbf{Idfor} C)$ iff $\forall s, s' \in C^{\mathcal{I}}$ and $\langle s, t_i \rangle, \langle s', t'_i \rangle \in R_i^{\mathcal{I}} \forall 1 \leq i \leq n$, we have $t_i = t'_i \forall 1 \leq i \leq n$ then $s = s'$.

Intuitively, this definition indicates that two instances of a concept C never share the same participation in these n roles. The roles must be *simple* (see section 5) to ensure the decidability of the reasoning algorithm (?). The abstract, DL syntax and semantics of ICs are presented in table ???.

For example, in DL framework the constraint in example ??? will be represented as the following definition : $(hasFlag, onDate \mathbf{Idfor} NationHistory)$, where $hasFlag$ is the abstract role, $onDate$ is the concrete role, $NationHistory$ is the concept.

4.4. RDF Graphs

In Semantic Web environment, an OWL ontology is in fact an RDF graph, which is in turn a set of RDF triples. Hence it is necessary to relate specific abstract syntax ontologies with specific RDF/XML documents and their corresponding graphs. We provide a mapping from the abstract syntax for OWL-K to the exchange syntax, namely RDF/XML syntax. Since OWL-K is the extension of OWL-DL, it inherits the existing mapping for OWL-DL (see section 4 of (?)). We introduce here only the mapping for IC assertions (see table ??). So that our extension preserves the normative relationship between the abstract syntax and the exchange syntax.

The IC in example 1 is represented in the exchange syntax as follows :

```
<owl:ICAssertion rdf:ID = "NationHistoryIC">
  <owl:onClass rdf:resource = "NationHistory" />
  <owl:byProperty rdf:resource = "#hasFlag"/>
  <owl:byProperty rdf:resource = "#onDate"/>
</owl:ICAssertion>
```

5. Decidability of OWL-K

As mentioned above, the underpinning of OWL-K is the DL language $\mathcal{SHOIN}(\mathbf{D})$. In this section, we will show that OWL-K is decidable by showing $\mathcal{SHOIN}(\mathbf{D})$ -concept satisfiability w.r.t KBs. To decide on the $\mathcal{SHOIN}(\mathbf{D})$ -concept satisfiability problem, a DL reasoner must handle the concept satisfiability w.r.t restrictions set by ICs in a given KB. This problem is addressed by introducing new Tableau expansion rules with some refinement techniques.

5.1. $\mathcal{SHOIN}(\mathbf{D})$ Syntax and Semantics

First of all, we briefly introduce the language $\mathcal{SHOIN}(\mathbf{D})$. This language is an extension of $\mathcal{SHOIN}(\mathbf{D})$ by combining it with ICs. Note that in DLs, we call *domain of data values* the *concrete domain*.

Definition 5.1 ($\mathcal{SHOIN}(\mathbf{D})$ Syntax and Semantics) *Let $\mathbf{C}, \mathbf{R} = \mathbf{R}_A \cup \mathbf{R}_D$ be disjoint sets of concept, abstract and concrete role names ; $\mathbf{R}_A = \mathbf{R}_{tp} \cup \{r^- | r \in \mathbf{R}_{tp}\}$ where $\mathbf{R}_{tp} = \mathbf{R}_+ \cup \mathbf{R}_{Ap}$ be disjoint sets of transitive and primitive abstract role names, r^- be the inverse role of r . A role box \mathcal{R} is a finite set of role axioms that are either role inclusions, which are of the form $r \sqsubseteq s$ for $r, s \in \mathbf{R}_A$ or $r, s \in \mathbf{R}_D$, or transitivity axioms, which are of the form $\text{Trans}(r)$ for $r \in \mathbf{R}_A$. r is a simple role if for $\sqsubseteq_{\mathcal{R}}$ the transitive-reflexive closure of \sqsubseteq on \mathcal{R} and for each role s , $s \sqsubseteq_{\mathcal{R}} r$ implies $\text{Trans}(s) \notin \mathcal{R}$.*

A TBox \mathcal{T} is a finite set of $\mathcal{SHOIN}(\mathbf{D})$ -concepts such that each concept name $A \in \mathbf{C}$ is a concept, and, for C and D concepts, R an abstract role, U a concrete role,

Tableau 6. *SHOINK(D) Syntax and Semantics*

Construct name	Syntax	Semantics
abstract atomic role	r	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
concrete role	u	$u^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta_{\mathbf{D}}^{\mathcal{I}}$
transitive role	$r \in \mathbf{R}_+$	$r^{\mathcal{I}} = r^{\mathcal{I}+}$
inverse role	r^-	$\{\langle x, y \rangle \mid \langle y, x \rangle \in r^{\mathcal{I}}\}$
role hierarchy	$r \sqsubseteq s$	$r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$
atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
datatype	D	$D_{\mathbf{D}} \subseteq \Delta_{\mathbf{D}}^{\mathcal{I}}$
concrete value	v	$v^{\mathcal{I}} = v^{\mathbf{D}}$
negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
exists restriction	$\exists R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y. \langle x, y \rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
value restriction	$\forall R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y. \langle x, y \rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
concrete exists restriction	$\exists R.d$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y. \langle x, y \rangle \in R^{\mathcal{I}} \wedge y \in d^{\mathbf{D}}\}$
concrete value restriction	$\forall R.d$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y. \langle x, y \rangle \in R^{\mathcal{I}} \rightarrow y \in d^{\mathbf{D}}\}$
atleast restriction	$\geq nR$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y. \langle x, y \rangle \in R^{\mathcal{I}}\} \geq n\}$
atmost restriction	$\leq nR$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y. \langle x, y \rangle \in R^{\mathcal{I}}\} \leq n\}$
nominal	o	$\#\{o^{\mathcal{I}}\} = 1$
identification constraint	$(R_1, \dots, R_n \text{ Idfor } C)$	$\mathcal{I} \models (R_1, \dots, R_n \text{ Idfor } C)$ iff $\forall s, s' \in C^{\mathcal{I}}$ and $\langle s, t_i \rangle, \langle s', t'_i \rangle \in R_i^{\mathcal{I}} \forall 1 \leq i \leq n$, $t_i = t'_i \forall 1 \leq i \leq n$ then $s = s'$

S a simple role, and $d \in \mathbf{D}$ a concrete datatype in the set \mathbf{D} of concrete datatypes, complex concepts can be built using the operators shown in table ??.

A Kbox \mathcal{K} is a finite set of definitions that describe ICs for concepts in a KB, and that have the form and the interpretation defined in definition ??.

The semantics is given in terms of an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consisting of a non-empty domain $\Delta^{\mathcal{I}}$, disjoint from the concrete domain $\Delta_{\mathbf{D}}$, and a mapping $\cdot^{\mathcal{I}}$ that maps every concept and role according to table ?? ($\#$ denotes set cardinality).

An interpretation \mathcal{I} satisfies a role box \mathcal{R} iff $r^{\mathcal{I}} \subseteq s^{\mathcal{I}} \forall r \sqsubseteq s \in \mathcal{R}$ and $r^{\mathcal{I}} = (r^{\mathcal{I}})^+ \forall \text{Trans}(r) \in \mathcal{R}$. A SHOINK(D)-concept C is satisfiable w.r.t. a role box \mathcal{R} and a Kbox \mathcal{K} iff there is an interpretation \mathcal{I} with $C^{\mathcal{I}} \neq \emptyset$ that satisfies \mathcal{R} and \mathcal{K} . Such an interpretation is called a model of C w.r.t. \mathcal{R} and \mathcal{K} . A concept C is subsumed by a concept D w.r.t. \mathcal{R} and \mathcal{K} iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for each interpretation \mathcal{I} satisfying \mathcal{R} and \mathcal{K} . Two concepts are said to be equivalent (w.r.t. \mathcal{R}, \mathcal{K}) iff they mutually subsume each other (w.r.t. \mathcal{R}, \mathcal{K}).

5.2. SHOINK(D) Reasoning

The decidability of OWL-K is addressed by proposing a decision procedure for SHOINK(D). This is an extension of the Tableau algorithm introduced by Ian Hor-

rocks and Ulrike Sattler in (?) by adding new expansion rules to support ICs with some refinement techniques.

$\mathcal{SHOIN}(\mathbf{D})$ Tableau algorithm decides the satisfiability of a concept C (w.r.t. a KB) by trying to construct for it an abstraction of a model called a forest-shaped *completion graph*. In this graph, each edge represents a set of abstract or of concrete roles; the nodes representing concrete values are called *concrete nodes*; those representing individuals are called *abstract nodes* which consist of *nominal* and *blockable* nodes. The set of blockable nodes is disjoint from that of nominal nodes. Edges and nodes are labeled with what they stand for. If there is an edge from node x to node y then y is called the *successor* of x and x is called the *predecessor* of y . *Ancestor* is the transitive closure of predecessor and *descendant* is the transitive closure of successor. A node y is called an *R-successor* of a node x if y is a successor of x and S is a role in the label of the edge from x to y and $S \sqsubseteq R$. A node y is called a *R-neighbor* of a node x if y is an R-successor of x or if x is an $\text{Inv}(R)$ -successor of y .

The graph is initialized with a set of tree-like structures rooted in nominal nodes or in root node labeled $\{C\}$. The algorithm exhaustively applies tableau rules that decompose the syntactic structure of the concepts in node labels, either expanding node labels, adding new edges and nodes to the “trees”, or *merging* edges and nodes. The application of a rule effectively explicates constraints on the interpretation implied by the concepts to which the rule is applied. An attempt to build a model fails if a contradiction, a so-called *clash*, is generated; or is successful if no more rules can be applied and there is no clash.

Being one of Tableau algorithms for expressive DLs, $\mathcal{SHOIN}(\mathbf{D})$ tableau algorithm employs a cycle detection technique called *pair-wise blocking* (?) to ensure termination. The algorithm generates a finite completion graph that can be *unravalled* into an infinite tree model, and where a node may stand for infinitely many elements of the model.

To obtain a decidable extension, adding ICs to $\mathcal{SHOIN}(\mathbf{D})$ requires various extensions as shown below.

5.2.1. Provide the Best Solution

Attached with a concrete domain, a DL reasoner must receive the values passed from a *solution* of a concrete domain reasoner to be able to decide the satisfiability of a concept. A solution δ for a conjunction c is a function mapping variables in c to values in a concrete domain so that $\delta(v^{(j)}) \in d_j^{\mathbf{D}}$, where d_j s are names of datatype; $v^{(j)}$ s are variables in a graph and $c = \bigwedge d_j(v^{(j)})$. If a solution returns the same value for separate variables, these variables are *equal*. However, if an IC requires that these variables must be different, a conflict occurs. To avoid the conflict between values returned by a solution and ICs in Kbox, the best solution in a sense that the variables are associated differently as many as possible must be provided. This solution is thus effected for the following conjunction :

$$c_{\mathcal{K}} = c \wedge \bigwedge_{\forall (v,v') \in V_{ie}} \neq (v, v') \quad [1]$$

where V_{ie} is the set of variables which are not *equal*. Two variables v and v' are called equal iff $\delta(v) = \delta(v')$. We look for the largest V_{ie} so that a solution δ is found for $c_{\mathcal{K}}$.

5.2.2. Merging Concrete Nodes

We define in graphs *identifying* and *identified nodes*. The former is one of the components identifying the latter. Note that ICs apply both to abstract and concrete roles. So that the identifying nodes are either abstract or concrete ones. Hence, to facilitate reasoning with ICs, the new rules should be synchronized for both abstract and concrete nodes. Because of the similarity of nominal and concrete nodes in the graph, merging concrete nodes will help for this purpose. Therefore, without loss of tree-shaped graph property, we introduce the v -rule to merge *equal* concrete nodes. Two nodes x and y are equal iff they stand for the same element (i.e. an individual of a concept or a concrete value of a datatype). This rule makes use of the function Merge and Prune defined in (?). However we modify the Prune in order to conform with concrete domain applied for ICs (see (?)).

5.2.3. Avoid Undecidability by Pair-wise Blocking Technique

Using *pair-wise blocking* technique to ensure the termination, the algorithm will generate a finite graph if a cycle is detected. This graph can then be *unravalled* into an infinite tree model where a node may stand for an infinite number of elements of the model. As a consequence, the undecidability may occur if ICs are applied on such a node. To avoid this case, we base ourselves on the decisive properties and hence introduce the new rules as follows :

- 1) Merging concrete nodes as discussed above results in the following property :

Lemme 5.1 *In the graph constructed by the Tableau algorithm for a given $\mathcal{SHOIN}\mathcal{K}(D)$ -concept, one node has more than one predecessor only if it is a nominal or a concrete node.*

- 2) If in a graph there is a node identified by n others, then by the nature of ICs no other node has the same connections. So if there is such another node, these two nodes must be equal and should be merged.

- 3) Representing identified nodes with nominals prevents merging from breaking the tree structure. So if we have two blockable identified nodes to be merged such that they do not have the same predecessor and all of their identifying nodes are their successors, a new nominal is added to the labels of these two nodes. Otherwise one node must be the ancestor of the other. In this case, the descendant is merged to the ancestor. This idea is realized by the \mathcal{K} -rule.

Lemme 5.2 *Let s and s' be the blockable nodes satisfying an IC in a graph. Merging s and s' does not break the tree structure with the condition of merging as follows :*

- *If all identifying nodes of s and s' are their descendants and s and s' do not have the same ancestor then add new nominal $o_{\mathcal{K}}$ to the label $\mathcal{L}(s)$ of s and to the label $\mathcal{L}(s')$ of s' .*

- *otherwise merge the descendant to its ancestor.*

4) The number of elements of a concept is naturally bounded by the domain of the concept. Therefore for identified nodes in a blocking cycle, we assume that their concrete identifying nodes have different concrete values. So that the cycle is terminated when the concrete domains for the respective concrete identifying nodes are exhaustively exploited. Note that with the best solution proposed, for an infinite domain of a datatype, δ always finds different values for variables in a graph. So that in such a case, two nodes identified by the same set of concrete nodes never exist.

Besides, we introduce two more rules, \mathcal{K}_+ -rule and $\leq_{\mathcal{K}}$ -rule, to guarantee that the semantics of ICs are fully captured. The details of these new expansion rules can be seen in (?).

5.2.4. Conditions of Termination

Adding ICs arises a new condition of clash that the Tableau algorithm must check as follows :

Definition 5.2 (Clash) *For $s \neq s'$ there exists a concept C so that $C \in \mathcal{L}(s) \cap \mathcal{L}(s')$ and $(R_1, \dots, R_n \text{ Idfor } C) \in \mathcal{K}$, $R_i \in \mathcal{L}(\langle s, t_i \rangle) \cap \mathcal{L}(\langle s', t_i \rangle)$ for all $1 \leq i \leq n$.*

Level of nominal nodes. The levels of primitive nominal nodes and the nominal nodes having nominal neighbors are defined as in (?). Identified nodes may not have nominal neighbors. Thus identified nodes having no nominal neighbors will be set level 0. Levels of nominal nodes are used to set the priority of application of expansion rules on graphs. It is very important to prove the termination. When a node in a lower level is merged to another, the level of the latter node may be reduced because the function Merge preserves all the connections between the nominal nodes. The rules are applied to the nodes from the lower level upwards.

The priority of applying the rules. For the rules which are the same as those given for $SHOIQ$, we keep the applying priority as described in (?). The new rules for ICs are applied with the lowest priority. This priority strategy is crucial for the termination of the algorithm.

With the extensions above, the algorithm for $SHOINK(\mathbf{D})$ is proved to be decidable (?).

Theorem 5.1 *The $SHOINK(\mathbf{D})$ -concept satisfiability problem w.r.t a KB is decidable.*

Since OWL-K corresponds to $\mathcal{SHOIN}\mathcal{K}(\mathbf{D})$ DL, we have the following corollary.

Corollary 5.1 *The OWL-K concept satisfiability problem w.r.t a KB is decidable.*

According to Tobies (?), if a DL language provides the nominal constructor then KB-satisfiability can be polynomially reduced to satisfiability of Tboxes, Rboxes and Kboxes. So that we obtain the following theorem.

Theorem 5.2 *The KB-satisfiability problem of OWL-K is decidable.*

6. Conclusion and Perspectives

Although OWL-DL is quite expressive, it has a very serious limitation on representing ICs as discussed above. In this paper, we propose OWL-K, an extension of OWL-DL, that supports this kind of constraints. The design of OWL-K satisfies all the requirements posed by ICs as presented in the end of section ???. OWL-K is decidable because its underpinning DL, $\mathcal{SHOIN}\mathcal{K}(\mathbf{D})$, can be proved decidable. This DL is a combination of $\mathcal{SHOIN}(\mathbf{D})$ and ICs. Since $\mathcal{SHOIN}(\mathbf{D})$ is already known with the complexity of NExpTime-complete (?), it is possible that in the worst case, the decision procedure for $\mathcal{SHOIN}\mathcal{K}(\mathbf{D})$ is not terminated in practice. As a consequence, OWL-K has a difficult entailment problem. Therefore the language extension presented here is designed as detachable components. This mechanism is also flexible for users who do not want to use it by simply switching off the extension.

Representation of ICs in the Semantic Web plays an important role. It assists in the designing phase of ontologies and provides the capacity to handle the consistency of KBs. It also provides the capacity to resolve the problem of interoperability and semantic integration of heterogeneous data sources, particularly of relational data sources in the Web environment.

For the future work, we are planning to implement this mechanism to Pellet, an open-source OWL-DL reasoner ², to support OWL-K, and evaluate it with some ontologies. It is also interesting to study how to construct a methodology of integration of relational data sources on the Semantic Web by using ICs introduced in this paper.

2. <http://pellet.owldl.com/>