Classifying Ontologies

Fábio Kepler, Christian Paz-Trillo, Joselyto Riani, Márcio M. Ribeiro, Karina Valdivia-Delgado, Leliane Nunes de Barros and Renata Wassermann

Institute of Mathematics and Statistics,
University of São Paulo
{kepler,cpaz,joselyto,marciomr,kvd,leliane,renata}@ime.usp.br

Abstract. During the development of an ontology it may be important to know which is the logic underlying that particular ontology, so that the developer knows what the expected complexity of reasoning over it will be.
In this paper, we first present an ontology that describes several description logics and then two different classifiers that were implemented using our ontology. The tools classify the ontology according to the minimal description logic that covers all constructors used.

1 Introduction

Recently, there has been much interest in modeling domains by means of ontologies described in formal languages. These ontologies can be used by different applications and are used to share and store knowledge. With the popularization of the Semantic Web [BLHL01], a lot of effort was put into defining languages for representing ontologies. Since 2004, OWL [MvH04] is a recommendation of the World Wide Web Consortium (W3C). OWL is based on description logics [HPS04], and there is now a growing interest in developing tools for the semantic web which make use of the logical power of the language.

In this paper, we present a case study in the area of ontologies and applications. We have constructed an ontology about the domain of description logics. We describe several logics and the relation among them. Then we have built two tools that use our ontology in order to classify other ontologies in terms of their expressivity and underlying logics. One of the tools is a plugin for Protégé[1] and the other is a Library using Jena[2], which can be used independently of the user interface.

The description logics ontology (DL-ontology henceforth) was built initially to help us to find our ways through the literature. There are many different logics and acronyms and their expressivity and complexity varies widely, from the most simple logic $\mathcal{FL}^0$ [LB87], for which inference is polynomial to very expressive logics such as $\mathcal{SHOIN}(\mathcal{D})$, which is one of the logics behind OWL [HPS04]. When building an ontology, it is useful to know which is the underlying logic, so

[1] A graphical ontology editor – http://protege.stanford.edu/.
[2] A Java API for Semantic Web applications – http://jena.sourceforge.net

that the developer knows the expected computational complexity of reasoning with his ontology. In several cases, it is possible to reformulate parts of the ontology and decrease the complexity by using a less expressive logic. Our tools are intended to help the develop-per in this task.

The paper proceeds as follows: in the next section, we briefly introduce a family of description logics and their relation to the Web Ontology Language OWL. In Section 3, we describe the DL-ontology as well as the process of constructing it. In Section 4, we describe the two tools that were implemented to classify ontologies using the DL-ontology. We finish the paper with some conclusions and issues for future work.

2 Description Logics and Ontologies

2.1 Description Logics – DL

Description Logics (DLs) are a family of knowledge representation languages which can be used to represent the terminological knowledge of an application domain in a structured and formally well-understood way [BCM+03].

In a DL, a distinction is drawn between the so-called "TBox" (Terminological Box) and the "ABox" (Assertional Box). Basically, the TBox contains sentences describing concept hierarchies (i.e., relations between concepts) while the ABox contains instance sentences stating where in the hierarchy individuals belong (i.e., relations between individuals and concepts). *Concepts* denote sets of individuals, and *roles* denote binary relations between individuals.

A DL system also offers services that *reason* about the stored terminologies and assertions. Typical reasoning tasks for a terminology are to determine whether a description is *satisfiable* (non-contradictory), or whether one description is more general than another. Important problems for assertions are to find out whether they are *consistent*, i.e., whether they have a model, and whether they entail that a particular individual is an *instance* of a given concept description.

Atomic concepts and *atomic roles* are elementary descriptions. Complex descriptions can be built from them inductively with *constructors*.

A DL can be characterized by the constructors it provides. For example, the logic $\mathcal{ALC}$ [LB87] is the DL that can have the constructors $\sqcup, \sqcap, \neg, \forall$, and $\exists$. Table 1 shows the principal constructors.

The DL $\mathcal{AL}$ (Attributive Language) is a minimal DL of practical interest. Other DLs of this family are extensions of $\mathcal{AL}$. The sub-language of $\mathcal{AL}$ obtained by disallowing atomic negation is called $\mathcal{FL}^-$ and the sub-language of $\mathcal{FL}^-$ obtained by disallowing limited existential quantification is called $\mathcal{FL}^0$.

More expressive DLs can be obtained by extending less expressive ones. For example, the DL $\mathcal{ALCQ}$ extends $\mathcal{AL}$ with the constructors negation ($\mathcal{C}$) and qualified number restrictions ($\mathcal{Q}$). $\mathcal{ALCH}$ denotes the extension of $\mathcal{ALC}$ by role hierarchies.

Symbol		Name
$\sqcup$	$\mathcal{U}$	Union
$\sqcap$	$\mathcal{AL}$	Intersection
$\neg$	$\mathcal{C}$	Concept negation (complement)
	$\mathcal{A}$	Atomic negation
$\forall$	$\mathcal{AL}$	Value restriction
$\exists$	$\mathcal{E}$	Existential quantification
$\geq, \leq, =$	$\mathcal{Q}$	Qualified number restriction
	$\mathcal{F}$	Functional number restriction
	$\mathcal{N}$	Unqualified number restriction
I	$\mathcal{O}$	Nominals (instance enumeration)
	$\mathcal{I}$	Inverse roles
	$\mathcal{H}$	Role hierarchies
	$\mathcal{D}$	Data type restriction

Table 1. Main constructors.

In order to avoid very long names for expressive DLs the abbreviation $\mathcal{S}$ was introduced for $\mathcal{ALC}_{R^+}$, i.e., the DL that extends $\mathcal{ALC}$ by transitive roles. Some members of the $\mathcal{S}$-family are $\mathcal{SIN}$, $\mathcal{SHIF}$, and $\mathcal{SHOIQ}$. To prevent ambiguity, we mixed both notations in this work. Instead of using $\mathcal{ALCHIQ}$ or $\mathcal{SHIQ}$, we use $\mathcal{ALC}$-$\mathcal{SHIQ}$.

2.2 Web Ontology Language – OWL

OWL is an acronym for *Web Ontology Language*, a markup language for publishing and sharing data using ontologies on the Internet [MvH04]. OWL is a vocabulary extension of the *Resource Description Framework* (RDF)[3] and is derived from the DAML+OIL[4] web ontology language.

OWL provides three increasingly expressive sub-languages designed for use by specific communities of implementers and users.

- OWL Lite supports those users primarily needing a classification hierarchy and simple constraints.
- OWL DL supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computed) and decidability (all computations will finish in finite time).
- OWL Full is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees.

Each of these sub-languages is an extension of its simpler predecessor, both in what can be legally expressed and in what can be validly concluded. The following set of relations hold, while their inverses do not:

- Every legal OWL Lite ontology is a legal OWL DL ontology.

[3] http://www.w3.org/RDF/
[4] http://www.w3.org/TR/daml+oil-reference/

- Every legal OWL DL ontology is a legal OWL Full ontology.
- Every valid OWL Lite conclusion is a valid OWL DL conclusion.
- Every valid OWL DL conclusion is a valid OWL Full conclusion.

OWL DL is based on the description logic $\mathcal{ALC}$-$\mathcal{SHOIN}$-$\mathcal{D}$ and its subset OWL Lite is based on the less expressive logic $\mathcal{ALC}$-$\mathcal{SHIF}$-$\mathcal{D}$ [HPS04].

3 The DL-ontology

3.1 Ontology development methodology

To develop an ontology from scratch is a difficult task. Even though there is a number of new technologies to support this process, like advanced ontology editors [MFG+03], this activity is still considered an art. Thus, it is an open issue the proposal of good methodologies to help an ontology designer. is given A brief overview on the state of the art of ontology development methodologies, which is still considered a not mature enough area, can be found in [MIS95].

The basic problem with those methodologies is that most of the existing ones are mainly concerned with providing guidelines for the whole building process (very similar with the purposes of software engineering methodologies). However, guidelines such as those for class and individual identifications, called the *bottom-layer model* of an ontology [MIS95], are kind of neglected.

During the construction of the DL-ontology, the few steps recommended by some of those methodologies [LGPSS99] [GF95] in order to support the development of the *bottom-layer model* of an ontology, are described bellow:

- **make motivation scenarios to formulate competency questions to be answered by the built ontology**. Our first motivation scenario was to have a compilation of the expertise knowledge on description logics and its connection with the Semantic Web area. A second motivation was to construct a description logic classifier, so an ontology designer could have answers about an ontology o such as: What is the logic underlying the ontology o? Can a reasoner X be used on o? How can the ontology o be reformulated so the reasoner X can be used? Can the ontology o be used to answer questions of type α (i.e., involving a set of given constructors)?
- **concepts collection** and **concepts definitions**. These two phases were based on technical documents [BCM+03] [AvH04]. Although the concepts and definitions were taken from these documents, the ontology construction was a difficult task. In order for the DL-ontology to be able to characterize answers to the competency questions, it was necessary to make several decisions, as we describe in the rest of this section in detail. For instance, the decision of characterizing a particular logic as a class instead of an individual allowed us to define axioms for each logic related to the set of constructors it supports.

3.2 Ontology Concepts

The proposed ontology, called DL-ontology, is intended to represent description logics and its expressiveness. More specifically, we focus on classification of logics based on the logical constructors it uses. We also introduce concepts for detecting reasoners that would support such description logics. The main concepts used in the ontology are:

- **Axiom:** represents the axioms that a description logic can support. An axiom can be a *Concept axiom* or a *Role axiom*. Each allowed axiom is declared as an instance of Concept or Role axiom. In the DL-ontology we have asserted two concept axioms: *definition* and *inclusion*; and two role axioms: *role hierarchy* and *transitivity*.
- **Logical Constructor:** is a concept in the DL-ontology that includes all the concept and role constructors allowed in description logics. We can list for example: *negation*, *concept intersection* and *concept union*.
- **Complexity:** represents any complexity class that is interesting for the description logics domain. The complexity classes are declared as instances of this class.
- **Reasoner:** reasoners that are associated to the description logics that they support, i.e., that can be used to make inferences about them.
- **Description Logic:** a higher level class so that any description logic is declared as a concept subsumed by this concept. We will use $\mathcal{ALC}$ to describe how a logic is represented:
  - **Constructors and Axioms:** represents lists of constructors and axioms that a description logic supports. For instance, the logic $\mathcal{ALC}$ supports: *Atomic Negation*, *Value Restriction*, *Full Existential Qualification*, *Concept Union*, *Intersection* and *Complement*.
  - **Complexity Class:** characterizes the complexity class for each description logic. The complexity class for $\mathcal{ALC}$ is PSpaceComplete.
  - **Reasoners:** lists the reasoners that can be used with the description logic. Examples of reasoners for $\mathcal{ALC}$ are: Pellet [5], Racer [6] and Fact++ [7].

In this work, we have characterized the family of $\mathcal{ALC}$ Logics. However any other description logic can be characterized, just by listing their constructors and axioms and associating its complexity and the reasoners that support it.

3.3 Modeling the Ontology

In this section we intend to show the choices we have made when modeling an ontology about description logics. Before starting to write the ontology one should decide for what purpose it will be used. In our case we needed an ontology which could satisfy three purposes:

[5] http://www.mindswap.org/2003/pellet/
[6] http://www.racer-systems.com/
[7] http://owl.man.ac.uk/factplusplus/

- to define description logics;
- to infer which of the well know description logics best characterizes a certain ontology with a certain set of constructors; and
- to allow extensions (e.g. if we want to introduce new logics).

Our first idea was to describe a logic as an individual (an element of the ABox), however, by doing this it was not possible to infer if some logic is subsumed by another. We have then changed to define a logic as a class.

The properties that distinguish one description logic from another are the constructors and the axioms they support. Hence in the ontology we have defined a certain description logic as "the description logic which can not have other constructors besides the listed ones". For example, the logic $\mathcal{ALC}$ in the DL-ontology is described by:

$$\mathcal{ALC} \equiv \mathcal{DL} \sqcap \forall\, \texttt{supports\_constructor}(\{\sqcup, \sqcap, \neg, \forall, \exists\}) \ . \tag{1}$$

which is consistent with our definition of what distinguishes each description logic. Moreover this description has the advantage that an hierarchy of description logics can be inferred. This hierarchy shows that some description logics are subsumed by others and inherit their properties such as being supported by some reasoner.

Furthermore we have decided to split the constructors into classes in order to make the ontology easier to read and manipulate. We also have chosen to join the constructors, which normally appear together, in the same class. For example: we have defined a class "Qualified Number Restriction" as the union of the class "Number Restriction" with all the constructors that characterize the qualified number restriction (Definition 2).

$$\begin{aligned} \texttt{Qualified\_Number\_Restriction} \equiv\ & \texttt{Number\_Restriction} \\ & \sqcup\ \{\texttt{atleast\_N\_Q}, \texttt{atmost\_N\_Q}\} \ . \end{aligned} \tag{2}$$

Hence, it is possible to define the logic $\mathcal{ALCQ}$ as:

$$\begin{aligned} \mathcal{ALCQ} \equiv \mathcal{DL} \sqcap \forall\, \texttt{supports\_constructor}(\{\sqcup, \sqcap, \neg, \forall, \exists\} \\ \sqcup\ \texttt{Qualified\_Number\_Restriction}) \ . \end{aligned} \tag{3}$$

From the definitions 3 and 1 one can infer that the logic $\mathcal{ALCQ}$ subsumes the logic $\mathcal{ALC}$. Some useful information can be inferred from the hierarchy of description logics. For example, the reasoner RACER can be used to reason about the description logic $\mathcal{SHIQ}$, and since the description logic $\mathcal{ALC}$ is subsumed (in our ontology) by the description logic $\mathcal{SHIQ}$ this implies that $\mathcal{ALC}$ inherits the property "being supported by RACER".

The DLs currently present in our ontology are: $\mathcal{FL}^-$, $\mathcal{FL}^0$, $\mathcal{AL}$, $\mathcal{ALU}$, $\mathcal{ALE}$, $\mathcal{ALC}$, $\mathcal{ALCF}$, $\mathcal{ALCQ}$, $\mathcal{ALCH}$, $\mathcal{ALCN}$, $\mathcal{ALC}$-$\mathcal{S}$, $\mathcal{ALC}$-$\mathcal{SH}$, $\mathcal{ALC}$-$\mathcal{SHI}$, $\mathcal{ALC}$-$\mathcal{SHIF}$, $\mathcal{ALC}$-$\mathcal{SHIF}$-$\mathcal{D}$, $\mathcal{ALC}$-$\mathcal{SHIQ}$, $\mathcal{ALC}$-$\mathcal{SHIN}$, $\mathcal{ALC}$-$\mathcal{SHIN}$-$\mathcal{D}$, $\mathcal{ALC}$-$\mathcal{SHO}$, $\mathcal{ALC}$-$\mathcal{SHOQ}$, $\mathcal{ALC}$-$\mathcal{SHOI}$, $\mathcal{ALC}$-$\mathcal{SHOIQ}$, $\mathcal{ALC}$-$\mathcal{SHOIQ}$-$\mathcal{D}$, $\mathcal{ALC}$-$\mathcal{SHON}$, $\mathcal{ALC}$-$\mathcal{SHOIN}$, $\mathcal{ALC}$-$\mathcal{SHOIN}$-$\mathcal{D}$.

3.4 The Ontology Complexity

The DL-ontology can be represented in the description logic $\mathcal{SOIF}$-$\mathcal{D}$, a sub-logic of $\mathcal{SHOIN}$-$\mathcal{D}$, since we have used the concept constructors union, nominals, value restriction and existential quantification, the transitive and inverse role axioms as well as data type restrictions.

Although we did not face any major performance issue when doing tests with our ontology, its logic happens to be very expressive and, consequently, complex to reason with in the general case [Lut04]. We conjecture that it is possible to devise an alternative ontology much less complex that still fulfills our requirements. As a matter of fact, in this study case we have not spent much attention to performance issues related to reasoning tasks with our ontology. This is mainly because we were most concerned with the expressive perspective (how to express things in the ontology) than the practical one (how fast would be the reasoning with it).

One can try to reduce an ontology complexity by removing some of the constructors used. Although this is not always possible, in some cases it can be done with minor changes in the ontology meaning without compromising its usability. For instance, the DL-ontology includes nominals, which is a well known 'killer' for current DL reasoners [HMW05]. We plan to investigate if it is possible to avoid such constructors in our ontology. In the following, we will sketch a preliminary idea of how to do that by modifying the DL-ontology (described in section 3.3) that can also be applied to any other ontology.

In order to eliminate the use of nominals, we will make some changes to the DL-ontology yielding a new ontology which we will call *the adapted DL-ontology*. The basic technique is to create a primitive concept for each possible logical constructor. Each of these primitive concepts will be called a *single constructor concept*. Note that, in our intended interpretation, each single constructor concept is a singleton whose instance represents a distinct constructor. We restrict the constructors range for each logic family, with the `supports_constructor` role, using the union of the respective single constructor concepts instead of nominals. For example, in the adapted ontology, the $\mathcal{ALC}$ logic family would be defined as

$$\mathcal{ALC} \equiv \mathcal{DL} \sqcap \forall \texttt{supports\_constructor}(U, I, C, UQ, EQ) \ . \tag{4}$$

where U, I, C, UQ, and EQ are single constructor concepts representing respectively (in our intended interpretation) the logical constructors union, intersection, complement, universal quantification and existential quantification.

Some preliminary tests with the adapted DL-ontology have shown that it does the same classifications as the DL-ontology. However, this must be carefully investigated because, although in our intended interpretation each constructor primitive concept is a singleton, there exists other possible interpretations. Therefore, the adapted DL-ontology is not logically equivalent to the original DL-ontology. The reason for that is that in the DL-ontology each constructor is an individual and in the adapted one they are primitive concepts. We could try to reduce this 'gap' by making some assertions in the adapted ontology. For instance, we could assert disjointness between each pair of single constructor

concepts. However, ultimately, it seems that the only way to express the individuality of each logical constructor is by the use of nominals since we cannot restrict the cardinality of a concept in description logics. On the other hand, the question that should be asked is if we really need to assert constructors individuality in order to fulfill the requirements of our ontology. This is not a trivial matter and we plan to address it in future work as well as what other constructors can be removed from our ontology yielding a less complex one which still fulfill our requirements.

4 Using the DL-ontology

We have implemented a plugin for Protégé and a Jena-Based Library for ontologies classification. They are both examples of applications that use the DL-ontology to classify ontologies in general. They require a DIG[8] and were tested with the Pellet reasoner.

4.1 Protégé plugin for Ontology Classification

The Protégé plugin OntoClassDL uses the Protégé OWL API[9], that is an open-source Java library for Web ontology and RDFS languages. The classes and methods provided by the API were used to implement the plugin that is able to access the DL-ontology to classify any other ontology.

In order to understand how the Protégé plugin for ontology classification works, we will use it to classify our own proposed ontology, i.e. the DL-ontology that has to be opened in the Protégé editor.

In the plugin implementation, the classOntoClassDL declares that an ontology class L is a subclass of the DL_Logic asserting the constructors and axioms it supports. Then, the DIG reasoner is called and the minimal description logic L' that includes all L logic constructors is returned. The reasoners that can make inference with that description logic are listed and the complexity class for that logic is also showed (Figure 1 shows a screenshot). In the example, the DL-ontology is classified as $\mathcal{ALC}$-$\mathcal{SHIF}$-$\mathcal{D}$, which is the description logic that includes all L logic constructors.

Suppose that we want to convert the DL-ontology into a DL with less complexity. The plugin finds in the inferred model a sub-logic L'' that includes some L logic constructors and shows the L'' features that are not used. In our example, the plugin finds one $\mathcal{ALC}$-$\mathcal{SHIF}$-$\mathcal{D}$ sub-logic, $\mathcal{ALC}$-$\mathcal{SHIF}$.

4.2 Jena-Based Library for Ontology Classification

We have also implemented a Jena-Based library for ontology classification that can be included in other applications independent of any user interface. This

[8] *Description Logic Interface* - a specification for a simple API to DL reasoners [BMC03]

[9] http://protege.stanford.edu/plugins/owl/api/

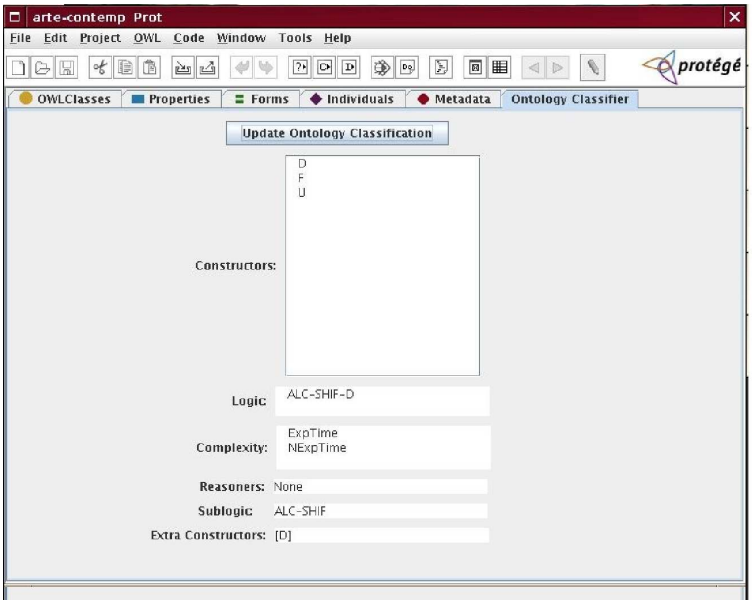

Fig. 1. OntoClassDL plugin

library was developed considering that the DL-ontology can be modified in order to be improved and extended.

We implement a Java class for every concept implemented in the ontology (See 3.2). The idea behind this library is that obtaining any property of an ontology concept can be seen as a query in SPARQL[10], so the main logic for classifying is entirely expressed in RDF queries and stored in files avoiding to be hard-coded inside the application.

In Figure 4.2 the process of classifying a user ontology is schematized. An application may pass a user ontology to the library to be classified. The library will create a L logic as a subclass of DL_Logic and will assert the constructors it supports. Then a DIG reasoner is called to classify the ontology. This will update the model that Jena keeps about the DL-ontology by creating instances of the ontology that is currently being classified. Queries in SPARQL were implemented for extracting the information from the library. Examples of such queries are: *What is the minimal logic that supports all the constructors of L? Which constructors could be avoided to obtain a simpler logic? What are the reasoners that can be used to reason over that logic?*

[10] SPARQL is a protocol and a query language for easy access to RDF graphs. It is candidate for being a W3C Recommendation. http://www.w3.org/TR/rdf-sparql-query

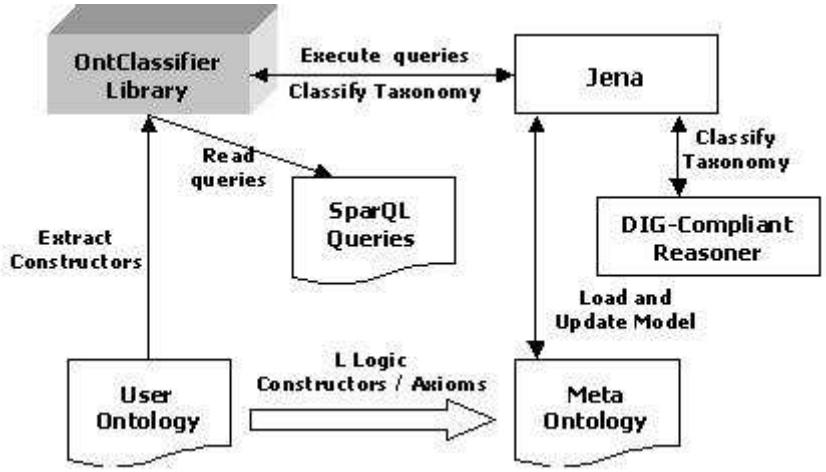

Fig. 2. The Classifier Library

4.3 Examples

The plugin and the Jena-Based Library were tested with three different versions of the Contemporary Art Ontology used in OnAIR[11] [PTWB05] and with the OWL-S [MPM+04] Service ontology [12].

The results are shown in Table 2. There are three versions of the Contemporary Art Ontology with varying syntactical characteristics in order to fall into less complex OWL sub-languages. The Data Type Restriction (D) makes them sub-logics of $\mathcal{ALC}$-$\mathcal{SHIF}$-$\mathcal{D}$. There exists logics that can fit better to the exact set of constructors used by these ontologies, but it is part of future work to extend the DL-ontology to deal with them.

Notice that the description logic ontology, that is centered in constructors, is not good to infer if DL-ontology is OWL-Full or OWL-DL. Because what distinguishes an ontology as being OWL-Full or OWL-DL is not only the constructors it can support.

The results with the Protégé plugin are the same as the ones found by the Jena-Based Library, the only difference is that the last provides all the sub logics L'' and for each one the L'' features not in use.

5 Conclusions and Future Work

In this paper we have presented an ontology for describing the domain of description logics, the DL-ontology, and two application tools that can classify an

[11] A query-based video retrieval system (http://bibo.incubadora.fapesp.br/portal/OnAIR/).
[12] The Service ontology provides a simple means of organizing the parts of a Web service description (http://www.daml.org/services/owl-s/1.1/Service.owl)

Ontology	Constructors	Logic	Reasoners	Complexity	Sub Logic	Non Sub Logic features used
$art - full$	D F U	ALC-SHIF-D	None	ExpTime NExpTime	ALC-SHIF	D
$art - dl$	D F U	ALC-SHIF-D	None	ExpTime NExpTime	ALC-SHIF	D
$art - lite$	D	ALC-SHIF-D	None	ExpTime NExpTime	ALC-SHIF	D
$OWL - S\_service$	I N	ALC-SHIN	FaCT FaCT++ Pellet Racer	ExpTime NExpTime	ALC-SHIF	N

Table 2. Results with the Contemporary art ontology and the $OWL - S\_service$ ontology

ontology regarding its expressivity and complexity of reasoning based on the DL-ontology.

This was a case study for the use of the technology that appeared recently around the semantic web. The DL-ontology was developed in OWL, using Protégé. One of the classifiers is a plugin for Protégé, the other one is a library based on Jena. Both have to connect to a reasoner using DIG.

The idea of having an ontology classifier as proposed here can be viewed as an important tool for novice ontology designers and therefore to be adopted as a fundamental step of any ontology development methodology. Its main utility is to guide a designer during the ontology modeling, to reformulate parts of the ontology in order to decrease its complexity by using, whenever possible, a less expressive logic.

Future work includes extending the tools to classify ontologies also regarding the fragment of OWL which can express them and better studying constructions to rewrite pieces of the ontology in order to reduce its complexity.

The ontology and the tools developed are available at
`http://bibo.incubadora.fapesp.br/portal/OntologiesClassification/`.

We would like to thank André Casado Castaño, Andréia Machion and George Henrique Silva for helping with the development of the tools.

References

[AvH04] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer*. MIT Press, Cambridge, MA, USA, 2004.

[BCM+03] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, May 2001.

[BMC03] S. Bechhofer, R. Moller, and P. Crowther. The DIG Description Logic Interface. In *Proc. of the International Workshop on Description Logics*, 2003.

[GF95] M. Grüninger and M. Fox. Methodology for the design and evaluation of ontologies. In *IJCAI'95, Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995.

[HMW05] Volker Haarslev, Ralf Möller, and Michael Wessel. Description logic inference technology: Lessions learned in the trenches. In *Description Logics*, 2005.

[HPS04] Ian Horrocks and Peter Patel-Schneider. Reducing OWL entailment to description logic satisfiability. *Journal of Web Semantics*, 1(4):345–357, 2004.

[LB87] Hector J. Levesque and Ronald J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3:78–93, 1987.

[LGPSS99] Mariano Fernandez Lopez, Asunción Gomez-Perez, Juan Pazos Sierra, and Alejandro Pazos Sierra. Building a chemical ontology using methontology and the ontology design environment. *IEEE Intelligent Systems*, 14(1):37–46, 1999.

[Lut04] Carsten Lutz. An improved NExpTime-hardness result for the description logic $\mathcal{ALC}$ extended with inverse roles, nominals, and counting. Technical Report LTCS-04-07, Inst. for Theoretical Computer Science, Dresden University of Technology, 2004. http://lat.inf.tu-dresden.de/research/reports.html.

[MFG+03] M. A. Muzen, R. W. Fergerson, W. E. Grosso, M. Crubezy, H. Heriksson, N. F. Noy, and S. W. Tu. Protégé: An environment for knowledge based systems development. *International Journal of Human-Computer Interaction*, 58(1):89–123, 2003.

[MIS95] R. Mizoguchi, M. Ikeda, and K. Seta. Ontology for modeling the world from problem solving perspectives. In *IJCAI'95, Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995.

[MPM+04] David L. Martin, Massimo Paolucci, Sheila A. McIlraith, Mark H. Burstein, Drew V. McDermott, Deborah L. McGuinness, Bijan Parsia, Terry R. Payne, Marta Sabou, Monika Solanki, Naveen Srinivasan, and Katia P. Sycara. Bringing semantics to web services: The OWL-S approach. In *Semantic Web Services and Web Process Composition, First International Workshop, (SWSWPC)*, volume 3387 of *LNCS*, pages 26–42. Springer, 2004.

[MvH04] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language overview. W3c recommendation, World Wide Web Consortium, February 2004.

[PTWB05] Christian Paz-Trillo, Renata Wassermann, and Paula P. Braga. An information retrieval application using ontologies. *Journal of the Brazilian Computer Society*, 11(2):17–31, 2005.