OWL-Eu: Adding Customised Datatypes into OWL

Jeff Z. Pan and Ian Horrocks

{pan,horrocks}@cs.man.ac.uk

School of Computer Science, University of Manchester, UK

Abstract. Although OWL is rather expressive, it has a very serious limitation on datatypes; i.e., it does not support customised datatypes. It has been pointed out that many potential users will not adopt OWL unless this limitation is overcome. Accordingly, the Semantic Web Best Practices and Development Working Group sets up a task force to address this issue. This paper makes the following two contributions: (i) it provides a brief summary of OWL-related datatype formalisms, and (ii) it provides a decidable extension of OWL DL, called OWL-Eu, that supports customised datatypes.

1 Introduction

The OWL Web Ontology Language [3] is a W3C recommendation for expressing ontologies in the Semantic Web. Datatype support [16, 17] is one of the most useful features OWL is expected to provide, and has brought extensive discussions in the RDF-Logic mailing list [18] and Semantic Web Best Practices mailing list [20]. Although OWL adds considerable expressive power to the Semantic Web, the OWL datatype formalism (or simply *OWL datatyping*) is much too weak for many applications; in particular, OWL datatyping does not provide a general framework for customised datatypes,[1] such as XML Schema derived datatypes.

It has been pointed out that many potential users will not adopt OWL unless this limitation is overcome [19], as it is often necessary to enable users to define their own datatypes and datatype predicates for their ontologies and applications. For instance, when using a computer sales ontology, a user may need to describe a PC with memory size greater than or equal to 512Mb, unit price less than 700 pounds and delivery date earlier than 15/03/2004. In this context, 'greater than or equal to 512', 'less than 700' and 'earlier than 15/12/2004' can be seen as customised datatypes, with the base datatypes being integer, integer and date, respectively.

After reviewing the design of OWL, and the needs of various applications and (potential) users, the following requirements for an extension to OWL DL have been identified:

1. It should provide customised datatypes; therefore, it should be based on a datatype formalism which is compatible with OWL datatyping, provides facilities to construct customised datatypes and, most importantly, guarantees the computerability of the kinds of customised datatypes it supports.

[1] A widely discussed example would be the 'BigWheel' example discussed in, e.g., `http://lists.w3.org/Archives/Public/public-swbp-wg/2004Apr/0061.html`.

2. It should overcome other important limitations of OWL datatyping, such as the absence of negated datatypes and the un-intuitive semantics for unsupported datatypes (which will be further explained in Section 4).
3. It should satisfy the *small extension requirement*, which is two folded: on the one hand, the extension should be a substantial and necessary extension that overcomes the above mentioned limitations of OWL datatyping; on the other hand, following W3C's 'one *small* step at a time' strategy, it should only be as large as is necessary in order to satisfy the requirements.
4. It should be a decidable extension of OWL DL.

This paper makes two main contributions. Firstly, it provides an overview of relevant (to OWL) datatype formalisms, namely those of XML, RDF and OWL itself. Secondly, and most importantly, it presents an extension of OWL DL,[2] called OWL-Eu (OWL with unary datatype Expressions), which satisfies the above requirements.

The rest of the paper is organised as follows. Section 2 briefly introduces the OWL Web Ontology Language. Section 3 describes OWL-related datatype formalisms. Section 4 summarises the limitations of OWL datatyping. Section 5 presents the OWL-Eu language, showing how it satisfies the above four requirements. Section 6 describes some related works, and Section 7 concludes the paper and suggests some future works.

2 An Overview of OWL

OWL is a standard (W3C recommendation) for expressing ontologies in the Semantic Web. The OWL language facilitates greater machine understandability of Web resources than that supported by RDFS by providing additional constructors for building class and property descriptions (vocabulary) and new axioms (constraints), along with a formal semantics. The OWL recommendation actually consists of three languages of increasing expressive power: OWL Lite, OWL DL and OWL Full. *OWL Lite* and *OWL DL* are, like DAML+OIL, basically very expressive Description Logics (DLs); they are almost[3] equivalent to the $\mathcal{SHIF}(\mathbf{D}^+)$ and $\mathcal{SHOIN}(\mathbf{D}^+)$ DLs. *OWL Full* provides the same set of constructors as OWL DL, but allows them to be used in an unconstrained way (in the style of RDF). It is easy to show that OWL Full is undecidable, because it does not impose restrictions on the use of transitive properties [10]; therefore, when we mention OWL in this paper, we usually mean OWL DL.

Let $\mathbf{C}$, $\mathbf{R_I}$, $\mathbf{R_D}$ and $\mathbf{I}$ be the sets of URIrefs that can be used to denote classes, *individual-valued* properties, *data-valued* properties and individuals respectively. An OWL DL *interpretation* is a tuple $\mathcal{I} = (\Delta^{\mathcal{I}}, \Delta_{\mathbf{D}}, \cdot^{\mathcal{I}}, \cdot^{\mathbf{D}})$ where the individual domain $\Delta^{\mathcal{I}}$ is a nonempty set of individuals, the datatype domain $\Delta_{\mathbf{D}}$ is a nonempty set of data values, $\cdot^{\mathcal{I}}$ is an individual interpretation function that maps

- each individual name $\mathtt{a} \in \mathbf{I}$ to an element $\mathtt{a}^{\mathcal{I}} \in \Delta^{\mathcal{I}}$,
- each concept name $\mathsf{CN} \in \mathbf{C}$ to a subset $\mathsf{CN}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$,
- each *individual-valued* property name $RN \in \mathbf{R_I}$ to a binary relation $RN^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and

[2] cf. Section 2 for the differences of three sub-languages of OWL.

[3] They also provide annotation properties, which Description Logics don't.

Abstract Syntax	DL Syntax	Semantics
ObjectProperty(R)	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
ObjectProperty(S inverseOf(R))	R^{-}	$(R^{-})^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

Table 1. OWL *individual-valued* property descriptions

– each *data-valued* property name $TN \in \mathbf{R_D}$ to a binary relation $TN^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta_{\mathbf{D}}$,

and $\cdot^{\mathbf{D}}$ is a datatype interpretation function. More details of $\Delta_{\mathbf{D}}$ and $\cdot^{\mathbf{D}}$ will be presented in Section 3.3.

The individual interpretation function can be extended to give semantics to class and *individual-valued* property descriptions shown in Tables 1 and 2, where $\mathsf{A} \in \mathbf{C}$ is a concept URIref, $C, C_1, \ldots, C_n$ are concept descriptions, $R \in \mathbf{R_I}$ is an *individual-valued* property URIref, $R_1, \ldots, R_n$ are *individual-valued* property descriptions and $\mathsf{o}, \mathsf{o}_1, \mathsf{o}_2 \in \mathbf{I}$ are individual URIrefs, u is a data range (cf. Definition 8), $T \in \mathbf{R_D}$ is a *data-valued* property and $\sharp$ denotes cardinality.

An OWL DL ontology can be seen as a DL knowledge base [11], which consists of a set of *axioms*, including class axioms, property axioms and individual axioms.[4] Table 3 presents the abstract syntax, DL syntax and semantics of OWL axioms.

Abstract Syntax	DL Syntax	Semantics
Class(A)	A	$\mathsf{A}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Class(owl:Thing)	$\top$	$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$
Class(owl:Nothing)	$\bot$	$\bot^{\mathcal{I}} = \emptyset$
intersectionOf($C_1, C_2, \ldots$)	$C_1 \sqcap C_2$	$(C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
unionOf($C_1, C_2, \ldots$)	$C_1 \sqcup C_2$	$(C_1 \sqcup C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
complementOf(C)	$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
oneOf($\mathsf{o}_1, \mathsf{o}_2, \ldots$)	$\{\mathsf{o}_1\} \sqcup \{\mathsf{o}_2\}$	$(\{\mathsf{o}_1\} \sqcup \{\mathsf{o}_2\})^{\mathcal{I}} = \{\mathsf{o}_1{}^{\mathcal{I}}, \mathsf{o}_2{}^{\mathcal{I}}\}$
restriction(R someValuesFrom(C))	$\exists R.C$	$(\exists R.C)^{\mathcal{I}} = \{x \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
restriction(R allValuesFrom(C))	$\forall R.C$	$(\forall R.C)^{\mathcal{I}} = \{x \mid \forall y.\langle x, y\rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
restriction(R hasValue(o))	$\exists R.\{\mathsf{o}\}$	$(\exists R.\{\mathsf{o}\})^{\mathcal{I}} = \{x \mid \langle x, \mathsf{o}^{\mathcal{I}}\rangle \in R^{\mathcal{I}}\}$
restriction(R minCardinality(m))	$\geqslant mR$	$(\geqslant mR)^{\mathcal{I}} = \{x \mid \sharp\{y.\langle x, y\rangle \in R^{\mathcal{I}}\} \geq m\}$
restriction(R maxCardinality(m))	$\leqslant mR$	$(\leqslant mR)^{\mathcal{I}} = \{x \mid \sharp\{y.\langle x, y\rangle \in R^{\mathcal{I}}\} \leq m\}$
restriction(T someValuesFrom(u))	$\exists T.u$	$(\exists T.u)^{\mathcal{I}} = \{x \mid \exists t.\langle x, t\rangle \in T^{\mathcal{I}} \wedge t \in u^{\mathbf{D}}\}$
restriction(T allValuesFrom(u))	$\forall T.u$	$(\forall T.u)^{\mathcal{I}} = \{x \mid \exists t.\langle x, t\rangle \in T^{\mathcal{I}} \rightarrow t \in u^{\mathbf{D}}\}$
restriction(T hasValue(w))	$\exists T.\{w\}$	$(\exists T.\{w\})^{\mathcal{I}} = \{x \mid \langle x, w^{\mathbf{D}}\rangle \in T^{\mathcal{I}}\}$
restriction(T minCardinality(m))	$\geqslant mT$	$(\geqslant mT)^{\mathcal{I}} = \{x \mid \sharp\{t \mid \langle x, t\rangle \in T^{\mathcal{I}}\} \geq m\}$
restriction(T maxCardinality(m))	$\leqslant mT$	$(\leqslant mT)^{\mathcal{I}} = \{x \mid \sharp\{t \mid \langle x, t\rangle \in T^{\mathcal{I}}\} \leq m\}$

Table 2. OWL class descriptions

[4] Individual axioms are also called *facts*.

Abstract Syntax	DL Syntax	Semantics
Class(A partial $C_1 \ \ldots \ C_n$)	$\mathsf{A} \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	$\mathsf{A}^{\mathcal{I}} \subseteq C_1^{\mathcal{I}} \cap \ldots \cap C_n^{\mathcal{I}}$
Class(A complete $C_1 \ \ldots \ C_n$)	$\mathsf{A} \equiv C_1 \sqcap \ldots \sqcap C_n$	$\mathsf{A}^{\mathcal{I}} = C_1^{\mathcal{I}} \cap \ldots \cap C_n^{\mathcal{I}}$
EnumeratedClass(A $\mathsf{o}_1 \ \ldots \ \mathsf{o}_n$)	$\mathsf{A} \equiv \{\mathsf{o}_1\} \sqcup \ldots \sqcup \{\mathsf{o}_n\}$	$\mathsf{A}^{\mathcal{I}} = \{\mathsf{o}_1{}^{\mathcal{I}}, \ldots, \mathsf{o}_n^{\mathcal{I}}\}$
SubClassOf(C_1, C_2)	$C_1 \sqsubseteq C_2$	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
EquivalentClasses($C_1 \ \ldots \ C_n$)	$C_1 \equiv \ldots \equiv C_n$	$C_1^{\mathcal{I}} = \ldots = C_n^{\mathcal{I}}$
DisjointClasses($C_1 \ \ldots \ C_n$)	$C_i \sqsubseteq \neg C_j$, $(1 \leq i < j \leq n)$	$C_1^{\mathcal{I}} \cap C_n^{\mathcal{I}} = \emptyset$, $(1 \leq i < j \leq n)$
SubPropertyOf(R_1, R_2)	$R_1 \sqsubseteq R_2$	$R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$
EquivalentProperties($R_1 \ \ldots \ R_n$)	$R_1 \equiv \ldots \equiv R_n$	$R_1^{\mathcal{I}} = \ldots = R_n^{\mathcal{I}}$
ObjectProperty(R super(R_1) ... super(R_n)	$R \sqsubseteq R_i$	$R^{\mathcal{I}} \subseteq R_i^{\mathcal{I}}$
domain(C_1) ... domain(C_k)	$\geqslant 1R \sqsubseteq C_i$	$R^{\mathcal{I}} \subseteq C_i^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
range(C_1) ... range(C_h)	$\top \sqsubseteq \forall R.C_i$	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times C_i^{\mathcal{I}}$
[Symmetric]	$R \equiv R^-$	$R^{\mathcal{I}} = (R^-)^{\mathcal{I}}$
[Functional]	$\mathsf{Func}(R)$	$\{\langle x, y\rangle \mid \sharp\{y.\langle x, y\rangle \in R^{\mathcal{I}}\} \leq 1\}$
[InverseFunctional]	$\mathsf{Func}(R^-)$	$\{\langle x, y\rangle \mid \sharp\{y.\langle x, y\rangle \in (R^-)^{\mathcal{I}}\} \leq 1\}$
[Transitive])	$\mathsf{Trans}(R)$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^+$
AnnotationProperty(R)		
Individual(o type(C_1)...type(C_n)	$\mathsf{o} : C_i, 1 \leq i \leq n$	$\mathsf{o}^{\mathcal{I}} \in C_i^{\mathcal{I}}, 1 \leq i \leq n$
value(R_1, o_1)...value(R_n, o_n)	$\langle \mathsf{o}, \mathsf{o}_i\rangle : R_i, 1 \leq i \leq n$	$\langle \mathsf{o}^{\mathcal{I}}, \mathsf{o}_i^{\mathcal{I}}\rangle \in R_i^{\mathcal{I}}, 1 \leq i \leq n$
SameIndividual($\mathsf{o}_1 \ \ldots \ \mathsf{o}_n$)	$\mathsf{o}_1 = \ldots = \mathsf{o}_n$	$\mathsf{o}_1^{\mathcal{I}} = \ldots = \mathsf{o}_n^{\mathcal{I}}$
DifferentIndividuals($\mathsf{o}_1 \ \ldots \ \mathsf{o}_n$)	$\mathsf{o}_i \neq \mathsf{o}_j, 1 \leq i < j \leq n$	$\mathsf{o}_i^{\mathcal{I}} \neq \mathsf{o}_j^{\mathcal{I}}, 1 \leq i < j \leq n$

Table 3. OWL axioms

3 Datatype Formalisms

In this section we will provide a brief overview of the XML, RDF and OWL datatype formalisms.

3.1 XML Schema Datatypes

W3C XML Schema Part 2 [4] defines facilities for defining simple types to be used in XML Schema as well as other XML specifications.

Definition 1 An *XML Schema simple type* d is characterised by a value space, $V(d)$, which is a non-empty set, a lexical space, $L(d)$, which is a non-empty set of Unicode [6] strings, and a set of facets, $F(d)$, each of which characterizes a value space along independent axes or dimensions. ◇

XML Schema simple types are divided into disjoint built-in simple types and derived simple types. Derived datatypes can be defined by derivation from primitive or existing derived datatypes by the following three means:

- Derivation by *restriction*, i.e., by using facets on an existing type, so as to limit the number of possible values of the derived type.
- Derivation by *union*, i.e., to allow values from a list of simple types.
- Derivation by *list*, i.e., to define the list type of an existing simple type.

Example 1. The following is the definition of a derived simple type (of the base datatype xsd:integer) which restricts values to integers greater than or equal to 0 and

less than 150, using the facets minInclusive and maxExclusive.

```
<simpleType name = "humanAge">
  <restriction base = "xsd:integer">
    <minInclusive value = "0" />
    <maxExclusive value = "150" />
  </restriction>
</simpleType>
```
♢

3.2 Datatypes in RDF

According to [8], RDF allows the use of datatypes defined by any external type systems, e.g., the XML Schema type system, which conform to the following specification.

Definition 2 A *datatype* d is characterised by a lexical space, $L(d)$, which is an non-empty set of Unicode strings; a value space, $V(d)$, which is an non-empty set, and a total mapping $L2V(d)$ from the lexical space to the value space. ⋄

This specification allows the use of non-list XML Schema built-in simple types as datatypes in RDF, although some built-in XML Schema datatypes are problematic because they do not fit the RDF datatype model.[5] Furthermore, comparisons between Definition 1 and 2 show that RDF does not take XML Schema facets into account, which are essential to define derived simple types.

In RDF, data values are represented by literals.

Definition 3 All *literals* have a lexical form being a Unicode string. *Typed literals* are of the form "s"^^u, where s is a Unicode string, called the *lexical form* of the typed literal, and u is a datatype URI reference. *Plain literals* have a lexical form and optionally a *language tag* as defined by [1], normalised to lowercase. ⋄

Example 2. Boolean is a datatype with value space $\{true, false\}$, lexical space {"true", "false","1","0"} and lexical-to-value mapping {"true"$\mapsto$ $true$, "false"$\mapsto$ $false$, "1"$\mapsto$ $true$, "0"$\mapsto$ $false$}. "true"^^$\mathrm{xsd:boolean}$ is a typed literal, while "true" is a plain literal. ♢

The associations between datatype URI references (e.g., $\mathrm{xsd:boolean}$) and datatypes (e.g., boolean) can be provided by datatype maps defined as follows.

Definition 4 A *datatype map* $\mathbf{M}_d$ is a partial mapping from datatype URI references to datatypes. ⋄

Note that XML Schema derived simple types are *not* RDF datatypes because XML Schema provides no mechanism for using URI references to refer to derived simple types.

The semantics of RDF datatypes are defined in terms of $\mathbf{M}_d$-interpretations, which extend RDF-interpretations and RDFS-interpretations (cf. RDF Semantics [8]) with extra conditions for datatypes.

[5] Readers are referred to [8] for more details.

Definition 5 Given a datatype map $\mathbf{M}_d$, an *RDFS* $\mathbf{M}_d$*-interpretation* I of a vocabulary **V** (a set of URIrefs and plain literals) is any RDFS-interpretation of $\mathbf{V} \cup \{u \mid \exists\, d.\langle u, d\rangle \in \mathbf{M}_d\}$ which introduces

- a non-empty set **IR** of resources, called the *domain* (or *universe*) of I,
- a set **IP** (the RDF-interpretation requires **IP** to be a sub-set of **IR**) called the *set of properties* in I,
- a set **IC** (the RDFS-interpretation requires **IC** to be a sub-set of **IR**) called the *set of classes* in I, and
- a distinguished subset **LV** of **IR**, called the *set of literal values*, which contains all the plain literals in **V**,
- a mapping IS from URIrefs in **V** to **IR**,
- a mapping $IEXT$, called the *extension function*, from **IP** to the powerset of $\mathbf{IR} \times \mathbf{IR}$,
- a mapping $ICEXT$, called the *class extension function*, from **IC** to the set of subsets of **IR**,
- a mapping IL from typed literals in **V** into **IR**,

and satisfies the following extra conditions:

1. **LV** = $ICEXT(IS(\mathrm{rdfs{:}Literal}))$,
2. for each plain literal pl, $IL(pl) = pl$,
3. for each pair $\langle u, d\rangle \in \mathbf{M}_d$,
   (a) $ICEXT(d) = V(d) \subseteq \mathbf{LV}$,
   (b) there exist $d \in \mathbf{IR}$ s.t. $IS(u) = d$,
   (c) $IS(u) \in ICEXT(IS(\mathrm{rdfs{:}Datatype}))$,
   (d) for "s"^^$u' \in \mathbf{V}$,$IS(u') = d$, if $s \in L(d)$, then $IL(\text{"s"}\hat{}\hat{}u') = L2S(d)(s)$, otherwise, $IL(\text{"s"}\hat{}\hat{}u') \in \mathbf{IR} \setminus \mathbf{LV}$,
4. if $d \in ICEXT(IS(\mathrm{rdfs{:}Datatype}))$, then $\langle d, IS(\mathrm{rdfs{:}Literal})\rangle \in IEXT(\text{rdfs:subClassOf})$. ◇

According to Definition 5, **LV** is a subset of **IR**, i.e., literal values are resources. Condition 1 ensures that the class extension of $\mathrm{rdfs{:}Literal}$ is **LV**. Condition 2 ensures that the plain literals are interpreted as themselves. Condition 3a asserts that RDF(S) datatypes are classes (because datatypes are interpreted using the class extension function $ICEXT$), condition 3b ensures that there is a resource d for datatype d in $\mathbf{M}_d$, and condition 3c ensures that the class $\mathrm{rdfs{:}Datatype}$ contains the datatypes used in any satisfying $\mathbf{M}_d$-interpretation. Condition 3d explains why the range of IL is **IR** rather than **LV** (because, for "s"^^u, if $s \notin L(IS(\mathrm{u}))$, then $IL(\text{"s"}\hat{}\hat{}u) \notin \mathbf{LV}$); note that this is different from OWL datatypes (cf. Definition 9). Condition 4 requires that RDF(S) datatypes are *sub-classes* of $\mathrm{rdfs{:}Literal}$.

3.3 Datatypes in OWL

OWL datatyping adopts the RDF specification of datatypes and data values. It extends RDF datatyping by (i) allowing different OWL reasoners to provide different supported datatypes, and (ii) introducing the use of so called enumerated datatypes.

Definition 6 Given a datatype map $\mathbf{M}_d$, a datatype URI reference u is called a *supported datatype URI reference w.r.t.* $\mathbf{M}_d$ if there exists a datatype d s.t. $\mathbf{M}_d(u) = d$ (in this case, d is called a *supported datatype w.r.t.* $\mathbf{M}_d$); otherwise, u is called an *unsupported datatype URI reference w.r.t.* $\mathbf{M}_d$. ◇

Definition 7 Let $y_1, \ldots, y_n$ be typed literals. An *enumerated datatype* is of the form oneOf($y_1 \ \ldots \ y_n$). ◇

Definition 8 An *OWL data range* has one of the forms: (i) a datatype URI reference, (ii) an enumerated datatype, or (iii) rdf:Literal. ◇

The semantics of OWL DL datatypes are defined in terms of OWL datatype interpretations.

Definition 9 An *OWL datatype interpretation* w.r.t. to a datatype map $\mathbf{M}_d$ is a pair $(\Delta_{\mathbf{D}}, \cdot^{\mathbf{D}})$, where the datatype domain $\Delta_{\mathbf{D}} = \mathbf{PL} \cup \bigcup_{\text{for each supported datatype URIref u w.r.t. } \mathbf{M}_p} V(\mathbf{M}_p(u))$ ($\mathbf{PL}$ is the value space for plain literals, i.e., the union of the set of Unicode strings and the set of pairs of Unicode strings and language tags) and $\cdot^{\mathbf{D}}$ is a datatype interpretation function, which has to satisfy the following conditions:

1. $\text{rdfs:Literal}^{\mathbf{D}} = \Delta_{\mathbf{D}}$;
2. for each plain literal l, $l^{\mathbf{D}} = l \in \mathbf{PL}$;
3. for each supported datatype URIref u (let $d = \mathbf{M}_d(u)$):
   (a) $u^{\mathbf{D}} = V(d) \subseteq \Delta_{\mathbf{D}}$,
   (b) if $s \in L(d)$, then $(\text{“s”}\hat{}\hat{}u)^{\mathbf{D}} = L2V(d)(s)$,
   (c) if $s \notin L(d)$, then $(\text{“s”}\hat{}\hat{}u)^{\mathbf{D}}$ is not defined;
4. for each unsupported datatype URIref u, $u^{\mathbf{D}} \subseteq \Delta_{\mathbf{D}}$, and $(\text{“s”}\hat{}\hat{}u)^{\mathbf{D}} \in u^{\mathbf{D}}$.
5. each enumerated datatype oneOf($y_1 \ \ldots \ y_n$) is interpreted as $y_1^{\mathbf{D}} \cup \ldots \cup y_n^{\mathbf{D}}$. ◇

The above definition shows that OWL datatyping is similar to RDF datatyping, except that (i) RDF datatypes are classes, while OWL DL datatypes are not classes,[6] and (ii) in RDF ill-defined typed literals are interpreted as resources in $\mathbf{IR} \setminus \mathbf{LV}$, while in OWL DL the interpretation of ill-defined typed literals are undefined.

4 Limitations of OWL Datatyping

OWL datatyping has the following serious limitations, which discourage potential users from adopting OWL DL in their SW and ontology applications [15, 19].

1. OWL does not support customised datatypes (except enumerated datatypes). Firstly, XML Schema derived simple types are not OWL DL datatypes, because of the problem of datatype URI references for XML Schema derived simple types. Secondly, OWL does not provide a mechanism to tell which (customised) datatypes can be used together so that the language is still decidable.

[6] In fact, classes and datatypes in OWL DL use different interpretation functions; cf. Section 2.

2. OWL does not support negated datatypes. For example, 'all integers but 0', which is the relativised negation of the enumerated datatype oneOf("0"^^$\mathrm{xsd:integer}$), is not expressible in OWL. Moreover, negated datatypes are *necessary* in the negated normal form (NNF)[7] of datatype-related class descriptions in, e.g., DL tableaux algorithms.
3. An OWL DL datatype domain seriously restricts the interpretations of typed literals with unsupported datatype URIrefs. For example, given the datatype map $\mathbf{M}_{d1} = \{\mathrm{xsd:integer} \mapsto \mathit{integer}, \mathrm{xsd:string} \mapsto \mathit{string}\}$, "1.278e-3"^^$\mathrm{xsd:float}$ has to be interpreted as either an integer, a string or a string with a language tag, which is counter-intuitive.

5 OWL-Eu

This section presents OWL-Eu and elaborates how OWL-Eu satisfies the four requirements (listed in Section 1) in the following four sub-sections.

5.1 Supporting Customised Datatypes

OWL-Eu supports customised datatypes through unary datatype expressions based on unary datatype groups. Intuitively, an unary datatype group extends the OWL datatyping with a hierarchy of supported datatypes.[8]

Definition 10 A *unary datatype group* $\mathcal{G}$ is a tuple ($\mathbf{M}_d$,$\mathbf{B}$,dom), where $\mathbf{M}_d$ is the *datatype map* of $\mathcal{G}$, $\mathbf{B}$ is the set of *primitive base datatype* URI references in $\mathcal{G}$ and dom is the *declared domain function*. We call $\mathbf{S}$ the set of supported datatype URI references of $\mathcal{G}$, i.e., for each $u \in \mathbf{S}$, $\mathbf{M}_d(u)$ is defined; we require $\mathbf{B} \subseteq \mathbf{S}$. We assume that there exists a unary datatype URI reference $\mathrm{owlx:DatatypeBottom} \notin \mathbf{S}$. The declared domain function dom has the following properties: for each $u \in \mathbf{S}$, if $u \in \mathbf{B}$, $\mathsf{dom}(u) = u$; otherwise, $\mathsf{dom}(u) = v$, where $v \in \mathbf{B}$. $\diamond$

Definition 10 ensures that all the primitive base datatype URIrefs of $\mathcal{G}$ are supported ($\mathbf{B} \subseteq \mathbf{S}$) and that each supported datatype URIref relates to a primitive base datatype URIref through the declared domain function dom.

Example 3. $\mathcal{G}_1 = (\mathbf{M}_{d1}, \mathbf{B}_1, \mathsf{dom}_1)$ is a unary datatype group, where

- $\mathbf{M}_{d1} = \{\mathrm{xsd:integer} \mapsto \mathit{integer}, \mathrm{xsd:string} \mapsto \mathit{string}, \mathrm{xsd:nonNegativeInteger} \mapsto \geq_0, \mathrm{xsdx:integerLessThanN} \mapsto <_N\}$,
- $\mathbf{B}_1 = \{\mathrm{xsd:string}, \mathrm{xsd:integer}\}$, and
- $\mathsf{dom}_1 = \{\mathrm{xsd:integer} \mapsto \mathrm{xsd:integer}, \mathrm{xsd:string} \mapsto \mathrm{xsd:string}, \mathrm{xsd:nonNegativeInteger} \mapsto \mathrm{xsd:integer}, \mathrm{xsdx:integerLessThanN} \mapsto \mathrm{xsd:integer}\}$.

[7] A concept is in negation normal form iff negation is applied only to atomic concept names, nominals or datatypes.

[8] Note that in [15] datatype groups allow arbitrary datatype predicates, while here we consider only datatypes, which can be regarded as *unary* datatype predicates.

According to $\mathbf{M}_{d1}$, we have $\mathbf{S}_1 = \{\text{xsd:integer}, \text{xsd:string}, \text{xsd:nonNegativeInteger}, \text{xsdx:integerLessThanN}\}$, hence $\mathbf{B}_1 \subseteq \mathbf{S}_1$. Note that the value space of $<_{\text{N}}$ is

$$V(<_{\text{N}}) = \{i \in V(integer) \mid i < L2S(integer)(\text{N})\},$$

and by $<_{\text{N}}$ we mean there exists a supported datatype $<_{\text{N}}$ for each integer $L2S(integer)(\text{N})$. ♢

Based on a unary datatype group, OWL-Eu provides a formalism (called datatype expressions) for constructing customised datatypes using supported datatypes.

Definition 11 Let $\mathcal{G}$ be a unary datatype group. The set of *$\mathcal{G}$-unary datatype expressions* in abstract syntax (corresponding DL syntax can be found in Table 5.1 on page 10), abbreviated $\mathbf{Dexp}(\mathcal{G})$, is inductively defined as follows:

1. *atomic expressions*: if u is a datatype URIref, then $u \in \mathbf{Dexp}(\mathcal{G})$;
2. *relativised negated expressions*: if u is a datatype URIref, then $\texttt{not}(u) \in \mathbf{Dexp}(\mathcal{G})$;
3. *enumerated datatypes*: if $l_1, \ldots, l_n$ are literals, then $\texttt{oneOf}(l_1, \ldots, l_n) \in \mathbf{Dexp}(\mathcal{G})$;
4. *conjunctive expressions*: if $\{E_1, ..., E_n\} \subseteq \mathbf{Dexp}(\mathcal{G})$, then $\texttt{and}(E_1, ..., E_n) \in \mathbf{Dexp}(\mathcal{G})$;
5. *disjunctive expressions*: if $\{E_1, ..., E_n\} \subseteq \mathbf{Dexp}(\mathcal{G})$, then $\texttt{or}(E_1, ..., E_n) \in \mathbf{Dexp}(\mathcal{G})$. ⋄

Example 4. $\mathcal{G}$-unary datatype expressions can be used to represent XML Schema non-list simple types. Given the unary datatype group $\mathcal{G}_1$ presented in Example 3 (page 8),

- built-in XML Schema simple types $integer$, $string$, $nonNegativeInteger$ are supported datatypes in $\mathcal{G}_1$;
- the XML Schema derived simple type (using only one facet)

```
<simpleType name = "lessThan5">
  <restriction base = "xsd:integer">
    <maxExclusive value = "5"/>
  </restriction>
</simpleType>,
```

  i.e. $<_5$, is a supported datatype in $\mathcal{G}_1$;
- the XML Schema derived simple type (using more than one facet) "humanAge" presented in Example 1 (page 4) can be represented by the following conjunctive expression

$$\texttt{and}(\text{xsd:nonNegativeInteger}, \text{xsdx:integerLessThan150});$$

- the following XML Schema derived union simple type

```
<simpleType name = "cameraPrice">
  <union>
    <simpleType>
      <restriction base = "xsd:nonNegativeInteger">
        <maxExclusive value = "100000"/>
      </restriction>
    </simpleType>
    <simpleType>
      <restriction base = "xsd:string">
        <enumeration value = "low"/>
        <enumeration value = "medium"/>
        <enumeration value = "expensive"/>
      </restriction>
    </simpleType>
  </union>
<simpleType>
```

  can be represented by the following disjunctive expression

  or(
    and(xsd:nonNegativeInteger, xsdx:integerLessThan100000)
    oneOf("low"^^xsd:string,"medium"^^xsd:string, "expensive"^^xsd:string)
). ♢

Abstract Syntax	DL Syntax	Semantics
a datatype URIref u	u	$u^{\mathbf{D}}$
oneOf($l_1, \ldots, l_n$)	$\{l_1, \ldots, l_n\}$	$\{l_1^{\mathbf{D}}\} \cup \ldots \cup \{l_n^{\mathbf{D}}\}$
not(u)	$\overline{u}$	$(\mathbf{M}_d(u))^{\mathbf{D}} \setminus u^{\mathbf{D}}$ if $u \in \mathbf{S} \setminus \mathbf{B}$ $\Delta_{\mathbf{D}} \setminus u^{\mathbf{D}}$ otherwise
and($E_1, \ldots, E_n$)	$E_1 \wedge \ldots \wedge E_n$	$E_1^{\mathbf{D}} \cap \ldots \cap E_n^{\mathbf{D}}$
or(P, Q)	$E_1 \vee \ldots \vee E_n$	$E_1^{\mathbf{D}} \cup \ldots \cup E_n^{\mathbf{D}}$

Table 4. Syntax and semantics of datatype expressions (OWL-Eu data ranges)

Definition 12 A *datatype interpretation* $\mathcal{I}_{\mathbf{D}}$ of a unary datatype group $\mathcal{G} = (\mathbf{M}_d, \mathbf{B}, \mathsf{dom})$ is a pair $(\Delta_{\mathbf{D}}, \cdot^{\mathbf{D}})$, where $\Delta_{\mathbf{D}}$ (the datatype domain) is a non-empty set and $\cdot^{\mathbf{D}}$ is a datatype interpretation function, which has to satisfy the following conditions:

1. $(\mathrm{rdfs{:}Literal})^{\mathbf{D}} = \Delta_{\mathbf{D}}$ and $(\mathrm{owlx{:}DatatypeBottom})^{\mathbf{D}} = \emptyset$;
2. for each plain literal l, $l^{\mathbf{D}} = l \in \mathbf{PL}$ and $\mathbf{PL} \subseteq \Delta_{\mathbf{D}}$;[9]
3. for any two primitive base datatype URIrefs $u_1, u_2 \in \mathbf{B}$: $u_1^{\mathbf{D}} \cap u_2^{\mathbf{D}} = \emptyset$;
4. for each supported datatype URIref $u \in \mathbf{S}$ (let $d = \mathbf{M}_d(u)$):
   (a) $u^{\mathbf{D}} = V(d) \subseteq \Delta_{\mathbf{D}}$, $L(u) \subseteq L(\mathsf{dom}(u))$ and $L2S(u) \subseteq L2S(\mathsf{dom}(u))$;
   (b) if $s \in L(d)$, then ("s"^^u)$^{\mathbf{D}} = L2V(d)(s)$; otherwise, ("s"^^$u$)$^{\mathbf{D}}$ is not defined;
5. $\forall u \notin \mathbf{S}$, $u^{\mathbf{D}} \subseteq \Delta_{\mathbf{D}}$, and "v"^^$u \in u^{\mathbf{D}}$.

Moreover, we extend $\cdot^{\mathbf{D}}$ to $\mathcal{G}$ unary datatype expression as shown in Table 5.1 (page 10). Let E be a $\mathcal{G}$ unary datatype expression, the negation of E is of the form $\neg E$, which is interpreted as $\Delta_{\mathbf{D}} \setminus E^{\mathbf{D}}$. $\diamond$

In Definition 12, Condition 3 ensures that the value spaces of all primitive base datatypes are disjoint with each other. Condition 4a ensures that each supported datatype is a derived datatype of its primitive base datatype. Please note the difference between a relativised negated expression and the negation of a unary datatype expression: the former one is a kind of unary datatype expression, while the latter one is the form of negation of all kinds of unary datatype expressions.

Now we introduce the kind of basic reasoning mechanisms required for a unary datatype group.

Definition 13 Let $\mathbf{V}$ be a set of variables, $\mathcal{G} = (\mathbf{M}_d, \mathbf{B}, \mathsf{dom})$ a unary datatype group. A datatype conjunctions of $\mathcal{G}$ of the form

$$\mathscr{C} = \bigwedge_{j=1}^{k} u_j(v_j) \wedge \bigwedge_{i=1}^{l} \neq_i (v_1^{(i)}, v_2^{(i)}), \tag{1}$$

where the v_j are variables from $\mathbf{V}$, $v_1^{(i)}, v_2^{(i)}$ are variables appear in $\bigwedge_{j=1}^{k} u_j(v_j)$, u_j are datatype URI references from $\mathbf{S}$ and $\neq_i$ are the inequality predicates for primitive base datatypes $\mathbf{M}_d(\mathsf{dom}(u_i))$ where u_i appear in $\bigwedge_{j=1}^{k} u_j(v_j)$.

[9] $\mathbf{PL}$ is the value space for plain literals; cf. Definition 9 on page 7.

A predicate conjunction $\mathscr{C}$ is called *satisfiable* iff there exist an interpretation $(\Delta_{\mathbf{D}}, \cdot^{\mathbf{D}})$ of $\mathcal{G}$ and a function δ mapping the variables in $\mathscr{C}$ to data values in $\Delta_{\mathbf{D}}$ s.t. $\delta(v_j) \in u_j^{\mathbf{D}}$ (for all $1 \leq j \leq k$) and $\{\delta(v_1^{(i)}), \delta(v_2^{(i)})\} \subseteq u_i^{\mathbf{D}}$ and $\delta(v_1^{(i)}) \neq \delta(v_2^{(i)})$ (for all $1 \leq i \leq l$). Such a function δ is called a *solution* for $\mathscr{C}$ w.r.t. $(\Delta_{\mathbf{D}}, \cdot^{\mathbf{D}})$. $\diamond$

We end this section by elaborating the conditions that computable unary datatype groups require.

Definition 14 A unary datatype group $\mathcal{G}$ is *conforming* iff

1. for any $u \in \mathbf{S} \setminus \mathbf{B}$: there exist $u' \in \mathbf{S} \setminus \mathbf{B}$ such that $u'^{\mathbf{D}} = \overline{u}^{\mathbf{D}}$, and
2. the satisfiability problems for finite datatype conjunctions of the form (1) is decidable. $\diamond$

5.2 Small Extension: From OWL DL to OWL-Eu

In this section, we present a small extension of OWL DL, i.e., OWL-Eu. The underpinning DL of OWL-Eu is $\mathcal{SHOIN}(\mathcal{G}_1)$, i.e., the $\mathcal{SHOIN}$ DL combined with a unary datatype group $\mathcal{G}$ (1 for unary). Specifically, OWL-Eu (only) extends OWL data range (cf. Definition 8) to OWL-Eu data ranges defined as follows.

Definition 15 An *OWL-Eu data range* is a $\mathcal{G}$ unary datatype expression. Abstract (as well as DL) syntax and model-theoretic semantics of OWL-Eu data ranges are presented in Table 5.1 (page 10). $\diamond$

The consequence of the extension is that customised datatypes, represented by OWL-Eu data ranges, can be used in datatype exists restrictions ($\exists T.u$) and datatype value restrictions ($\forall T.u$), where T is a datatype property and u is an OWL-Eu data range (cf. Table 2 on page 3). Hence, this extension of OWL DL is as large as is necessary to support customised datatypes.

Example 5. PCs with memory size greater than or equal to 512 Mb and with price cheaper than 700 pounds can be represented in the following OWL-Eu concept description in DL syntax (cf. Table 5.1 on page 10):

$$\mathsf{PC} \sqcap \exists memorySizeInMb.\overline{<_{512}} \sqcap \exists priceInPound.\ <_{700},$$

where $\overline{<_{512}}$ is a relativised negated expression and $<_{700}$ is a supported datatype in $\mathcal{G}_1$. $\diamondsuit$

5.3 Decidability of OWL-Eu

Theorem 5.19 of [15] indicates that we can combine any decidable DL that provides the conjunction ($\sqcap$) and bottom ($\bot$) constructors with a conforming unary datatype group and the combined DL is still decidable. Therefore, OWL-Eu is decidable.

Theorem 1. *(Theorem 6.2 of [15]) The knowledge base satisfiability problem of OWL-Eu is decidable if the combined unary datatype group is conforming.*

5.4 Overcoming the Limitations of OWL Datatyping

This section summarises how OWL-Eu overcomes the limitations of OWL datatyping presented in Section 4. Firstly, OWL-Eu is a decidable extension (Theorem 1) of OWL DL that supports customised datatypes with unary datatype expressions (cf. Example 4). Secondly, Definition 12 defines the negations of datatype expressions and OWL-Eu provides relativised negated datatype expression (Definition 11). Thirdly, according to Definition 12, the datatype domain in an interpretation of a datatype group is a superset of (instead of equivalent to) the value spaces of primitive base datatypes and plain literals; hence, typed literals with unsupported predicates are interpreted more intuitively.

6 Related Work

The concrete domain approach [2, 14] provides a rigorous treatment of datatype predicates, rather than datatypes.[10] In the type system approach [12], datatypes are considered to be sufficiently structured by type systems; however, it does not specify how the derivation mechanism of a type system affects the set of datatypes **D**. [5] suggests some solutions to the problem of referring to an XML Schema user defined simple type with a URI reference; however, it does not address the computability issue of combining the $\mathcal{SHOIN}$ DL with customised datatypes.

7 Discussion

Although OWL is rather expressive, it has a very serious limitation on datatypes; i.e., it does not support customised datatypes. It has been pointed out that many potential users will not adopt OWL unless this limitation is overcome. Accordingly, the Semantic Web Best Practices and Development Working Group sets up a task force to address this issue. As discussed above, a solution for the problem should cover much more than just a standard way of referring to an XML Schema user defined simple type with a URI reference.

In this paper, we propose OWL-Eu, an extension of OWL DL that supports customised datatypes. The underpinning of OWL-Eu is the $\mathcal{SHOIN}(\mathcal{G}_1)$ DL, a combination of $\mathcal{SHOIN}$ and a unary datatype group. OWL-Eu is decidable if the combined unary datatype group is conforming; the conformability of a unary datatype group precisely specifies the conditions on the set of supported datatypes. OWL-Eu provides a general framework for integrating OWL DL with customised datatypes, such as XML Schema non-list simple types.

We have implemented a prototype extension of the FaCT [9] DL system to support TBox reasoning of the $\mathcal{SHIQ}(\mathcal{G}_1)$ DL, a sub-language of OWL-Eu. As for future work, we are planing to extend the DIG1.1 interface [7] to support OWL-Eu and to implement a Protégé [13] plug-in to support XML Schema non-list simple types, i.e. users should be able to define and/or import customised XML Schema non-list simple types based on a set of supported datatypes, and to exploit our prototype through the extended DIG interface.

[10] The reader is referred to Section 5.1.3 of [15] for detailed discussions on concrete domains.

Bibliography

[1] H. Alvestrand. Rfc 3066 - tags for the identification of languages. Technical report, IETF, Jan 2001. http://www.isi.edu/in-notes/rfc3066.txt.

[2] Franz Baader and Philipp Hanschke. A Schema for Integrating Concrete Domains into Concept Languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 452–457, 1991.

[3] Sean Bechhofer, Frank van Harmelen, James Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein eds. OWL Web Ontology Language Reference. http://www.w3.org/TR/owl-ref/, Feb 2004.

[4] Paul V. Biron and Ashok Malhotra. Extensible Markup Language (XML) Schema Part 2: Datatypes – W3C Recommendation 02 May 2001. Technical report, World Wide Web Consortium, 2001. http://www.w3.org/TR/xmlschema-2/.

[5] Jeremy J. Carroll and Jeff Z. Pan. XML Schema Datatypes in RDF and OWL. Technical report, W3C Semantic Web Best Practices and Development Group, Nov 2004. Editors' Draft, http://www.w3.org/2001/sw/BestPractices/XSCH/xsch-sw/.

[6] Unicode Consortium. *The Unicode Standard*. Addison-Wesley, 2000. ISBN 0-201-61633-5. version 3.

[7] DIG. SourceForge DIG Interface Project. http://sourceforge.net/projects/dig/, 2004.

[8] Patrick Hayes. RDF Semantics. Technical report, W3C, Feb 2004. W3C recommendation, http://www.w3.org/TR/rdf-mt/.

[9] I. Horrocks. Using an Expressive Description Logic: FaCT or Fiction? In *Proc. of KR'98*, pages 636–647, 1998.

[10] I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Expressive Description Logics. In *Proc. of Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in LNAI, pages 161–180, 1999.

[11] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[12] Ian Horrocks and Ulrike Sattler. Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 199–204, 2001.

[13] Holger Knublauch, Ray W. Fergerson, Natalya Fridman Noy, and Mark A. Musen. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In *International Semantic Web Conference*, pages 229–243, 2004.

[14] Carsten Lutz. *The Complexity of Reasoning with Concrete Domains*. PhD thesis, Teaching and Research Area for Theoretical Computer Science, RWTH Aachen, 2001.

[15] Jeff Z. Pan. *Description Logics: Reasoning Support for the Semantic Web*. PhD thesis, School of Computer Science, The University of Manchester, Oxford Rd, Manchester M13 9PL, UK, 2004.

[16] Jeff Z. Pan and Ian Horrocks. Extending Datatype Support in Web Ontology Reasoning. In *Proc. of the 2002 Int. Conference on Ontologies, Databases and Applications of SEmantics (ODBASE 2002)*, Oct 2002.

[17] Jeff Z. Pan and Ian Horrocks. Web Ontology Reasoning with Datatype Groups. In *Proc. of the 2003 International Semantic Web Conference (ISWC2003)*, pages 47–63, 2003.

[18] RDF-Logic Mailing List. http://lists.w3.org/archives/public/www-rdf-logic/. W3C Mailing List, starts from 2001.

[19] A. Rector. Re: [UNITS, OEP] FAQ : Constraints on data values range. Discussion in [20], Apr. 2004. http://lists.w3.org/Archives/Public/public-swbp-wg/2004Apr/0216.html.

[20] Semantic Web Best Practice and Development Working Group Mailing List. http://lists.w3.org/archives/public/public-swbp-wg/. W3C Mailing List, starts from 2004.