

The DIG Description Logic Interface: DIG/1.1

Sean Bechhofer
University of Manchester
Oxford Road
Manchester M13 9PL
seanb@cs.man.ac.uk

February 7, 2003

Abstract

This document describes a simple API for a general description logic system.

1 Introduction

Most description logic (DL) systems present the application programmer with a functional interface, often defined using a Lisp-like syntax. Such interfaces may be more or less complex, depending on the sophistication of the implemented system, and may be more or less compliant with a specification such as KRSS [11].

The Lisp style of the KRSS syntax reflects the fact that Lisp is still the most common implementation language for DLs. This can create considerable barriers to the use of DL systems by application developers, who often prefer other languages (in particular the currently ubiquitous Java), and who are becoming more accustomed to component based software development environments. This is of increasing importance given current interest in Web Services and service based architectures.

In such an environment, a DL might naturally be viewed as a self contained component, with implementation details, and even the precise location in which its code is being executed, being hidden from the application [3]. This approach has several advantages: the issue of implementation language is finessed; the API can be defined in some standard formalism intended for the purpose; a mechanism is provided for applications to communicate with the DL system, either locally or remotely; and alternative DL components can be substituted without affecting the application.

This approach was adopted in the CORBA-FaCT system [5], where a CORBA interface was defined for a description logic reasoner. This wrapping of the FaCT reasoner facilitated the successful use of the reasoner in applications such as OilEd [4] and ICOM [6].

Although useful, CORBA-FaCT suffered from a number of inadequacies. The concept language does not cover concrete domains, and as the interface was largely based on the FaCT reasoner (which did not support an A-box at the time), functionality relating to Aboxes was missing. In addition the identifiers allowed were closely tied to the underlying Lisp implementation (case insensitive strings of essentially alphanumeric characters). This introduces problems when trying to reason over languages such as

DAML+OIL where classes are referred to using URIs¹.

The RACER [8] system adopted a slightly different mechanism, providing a socket based interface for use by client applications. This again provides a language neutral API, but the lower-level interface places more onus on the client programmer.

As part of the activity of the Description Logic Implementation Group (DIG²) a new interface for DL systems is being defined.

The specification described here should be considered as a *Level 0* specification – in its current version it contains just enough functionality to enable tools such as OilEd [4] to use the reasoner. Later version of the specification will address issues such as stateful connections, transactions, reasoner preferences and so on. There may also be the possibility of extending the concept language.

There is nothing inherently new in this specification – it is effectively an XML Schema for a DL concept language along with ask/tell functionality. Along with the definition of this interface, however, there is a commitment from the implementors of the leading DL reasoners (FaCT [9], RACER [8] and Cerebra³) to provide implementations conforming to the specification. This will truly allow us to build plug and play applications where alternative reasoners can be seamlessly integrated into our systems.

2 Overview

A number of assumptions have been made for this initial specification.

- The specification is agnostic as to multiple client connections. Multi-threaded implementations of a reasoner may be provided, but no guarantees are made as to the semantics when clients attempt to simultaneously update and query.
- The connection to the reasoner is effectively stateless. Clients are not identified to the reasoner, thus the reasoner will not distinguish between clients and maintain any kind of consistency checking or record of which client is adding information or making requests. Conversely, a client has no way of ensuring that the reasoner has not been given additional information (such as additional axioms) since its last communication.
- There is no explicit classification request. The reasoner will decide when it is appropriate to, for example, build a classification hierarchy of concepts. This may happen after each TELL request, alternatively the reasoner may choose to defer the classification until absolutely necessary, or even when there is a lull in traffic.

The specification essentially consists of an XML Schema [12] describing the expressions of the concept language, the available tell and ask operations along with the expected responses and administrative information

The schema is described loosely in this document, but the latest up to date schema is available from:

<http://dl-web.man.ac.uk/dig/2003/02/dig.xsd>

¹Of course such problems are not insurmountable, but do provide barriers to the ease of use of the systems.

²<http://dl.kr.org/dig>

³<http://www.networkinference.com/>

3 Protocol

Level 0 uses HTTP [7] as the underlying transfer protocol. In this respect, we borrow from other initiatives such as SOAP⁴ and XML-RPC⁵ which have both built messaging protocols using XML on top of HTTP. The use of HTTP allows client (and server) developers to use existing libraries for implementation⁶. We are not strongly wedded to the use of HTTP as the underlying protocol. A richer mechanism may be adopted in the future. For this Level 0 specification, however, it will suffice.

3.1 Request

Clients communicate with a server through the use of HTTP POST requests. The body of the request must be an XML encoded message corresponding to a DIG request as described below. `Content-Type` is `text/xml`, and the `Content-Length` must be specified and must be correct.

The server will use the root element of the message body to determine the message type (i.e. identification, management, ask or tell).

3.2 Response

Unless there is a low-level error, the server should return 200 OK. As with requests, the `Content-Type` is `text/xml` and `Content-Length` must be present and correct. The body of the response must be an XML encoded message corresponding to a DIG response as described below.

3.3 Persistent Connections

The HTTP specification supports the use of persistent connections, which allow requests to be pipelined. This can allow a single TCP connection to be used for multiple requests without waiting for each response. It is envisaged that DIG reasoners implementing this specification will support persistent connections.

4 Reasoner Identification

An aspect lacking from the original CORBA-FaCT specification was the ability to identify which reasoner was actually behind the interface. This is particularly important when we may have a number of different reasoners supplying conforming interfaces.

4.1 Reasoner Capabilities

Ideally, we would expect all reasoners implementing the specification to support the entire concept language and tell/ask functionality. In reality, this is unlikely to be the case in the short term, and some reasoners may choose not to implement, for example, support for concrete domains. In order to cope with this, along with information regarding their identification, a reasoner should also supply details of the language which

⁴<http://www.w3.org/TR/SOAP/>

⁵<http://www.xmlrpc.com/>

⁶Although this should not be a prime motivation for the use of the protocol, building on top existing work is likely to improve the chances of the DIG Interface being used.

it supports. This will enable clients to decide whether or not the reasoner will be of use, or guide the clients as to the questions that they can ask of the reasoner.

Such “introspective” descriptions of tools and services are crucial to supporting dynamic component and service discovery as is being pursued in areas such as the Grid.

In the current specification this capability information is rather primitive, and essentially amounts to a list of the concept forming operators, tell assertions and queries supported. In the long term, it would be desirable to extend this, for example being able to represent constraints such as whether particular combinations of operation are allowed.

It is assumed that all reasoners will support primitive concepts and roles.

4.1.1 Unsupported Constructs

If a reasoner receives a message (either TELL or ASK) that contains any syntax that it does not understand, an `error` should be returned.

4.2 Identification Request

An IDENTIFIER request contains a single `<getIdentifier>` element (See Figure 1).

```
<?xml version="1.0" encoding="UTF-8"?>
<getIdentifier
  xmlns="http://dl.kr.org/dig/2003/02/lang"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://dl.kr.org/dig/2003/02/lang
  http://dl-web.man.ac.uk/dig/2003/02/dig.xsd" />
```

Figure 1: Reasoner Identification Request

The response to an IDENTIFIER request should consist of a single `<identifier>` element. This element includes a string identifying the reasoner along with its version number (which is itself a sequence of integers separated by the “.” character) and an optional explanatory message. In addition, a `<supports>` element describes the language and capabilities of the reasoner. Figure 2 shows a sample identification response from a reasoner. This reasoner supports a small subset of the language – conjunction, disjunction, negation and existential quantification. The only assertions (apart from the introduction of primitives) are concept implications, and only satisfiability tests can be asked⁷.

5 Knowledge Base management

A DIG reasoner can deal with multiple knowledge bases. URIs are used In order to identify the different knowledge bases. When a request is made to a reasoner to create a new knowledge base, the reasoner (if successful) will return to the client a URI which the client can then use to identify the knowledge base during TELL and ASK requests (see Sections 7 and 8). The URI is valid for that reasoner only – making a request to

⁷We would, of course, in general expect reasoners to support a much larger subset of the operations described in the DIG schema.

```

<?xml version="1.0" encoding="UTF-8"?>
<identifier
  xmlns="http://dl.kr.org/dig/2003/02/lang"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://dl.kr.org/dig/2003/02/lang
http://dl-web.man.ac.uk/dig/2003/02/dig.xsd"
  name="FaCT" version="13.0.4" message="FaCT reasoner running on
potato.cs.man.ac.uk (130.88.196.149)">
  <supports>
    <language>
      <and/>
      <or/>
      <not/>
      <some/>
    </language>
    <tell>
      <impliesc/>
    </tell>
    <ask>
      <satisfiable/>
    </ask>
  </supports>
</identifier>

```

Figure 2: Reasoner Identification

another reasoner with the same URI will result in an error. Different clients of the same reasoner, however, may be able to share knowledge bases by sharing URIs – however it is the clients’ responsibility to manage and coordinate this sharing.

There are two MANAGEMENT requests. A request for a new knowledge base consists of a single <newKB> element (See Figure 3).

```

<?xml version="1.0" encoding="UTF-8"?>
<newKB
  xmlns="http://dl.kr.org/dig/2003/02/lang"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://dl.kr.org/dig/2003/02/lang
http://dl-web.man.ac.uk/dig/2003/02/dig.xsd"/>

```

Figure 3: New Knowledge Base Request

The response to a new KB request should consist of a single <response> element. This element contains either a single <kb> element indicating the URI that the reasoner has allocated for the KB, or an <error> message indicating that an error occurred.

```

<?xml version="1.0" encoding="UTF-8"?>
<response
  xmlns="http://dl.kr.org/dig/2003/02/lang"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://dl.kr.org/dig/2003/02/lang
http://dl-web.man.ac.uk/dig/2003/02/dig.xsd">
  <kb uri="urn:uuid:abcdefgh-1234-1234-12345689ab"/>
</response>

```

Figure 4: Successful Knowledge Base Creation Response

Figure 4 shows a possible response from such a request. In this case, the server has created a new UUID to refer to the knowledge base. This URI should then be used during TELL and ASK requests made against the knowledge base.

Client can release a knowledge base through a request consisting of a single `<releaseKB>` element with an attribute specifying the KB to release. Once a knowledge base has been released, any requests made using the URI should result in an error. Figures 5 and 6 show example release requests and the response if the release was successful.

```
<?xml version="1.0" encoding="UTF-8"?>
<releaseKB
  xmlns="http://dl.kr.org/dig/2003/02/lang"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://dl.kr.org/dig/2003/02/lang
http://dl-web.man.ac.uk/dig/2003/02/dig.xsd"
  uri="urn:uuid:abcdefgh-1234-1234-12345689ab"/>
```

Figure 5: Release Knowledge Base Request

```
<?xml version="1.0" encoding="UTF-8"?>
<response
  xmlns="http://dl.kr.org/dig/2003/02/lang"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://dl.kr.org/dig/2003/02/lang
http://dl-web.man.ac.uk/dig/2003/02/dig.xsd">
  <ok/>
</response>
```

Figure 6: Successful Knowledge Base Release Response

6 Concept Language

DIG's concept language is based on $SHOIQD_n^-$, that is a description logic that includes the standard boolean concept operators (and, or, not), universal and existential restrictions, cardinality constraints, a role hierarchy, inverse roles, the one-of construct and concrete domains. $SHOIQD_n^-$ was chosen as it is rich enough to support reasoning over the DAML+OIL language. Reasoning for Semantic Web applications is seen as a prime use for DL reasoners in the near future.

This Level 0 Version 1.1 provides rather restricted support for concrete domains. Integers and strings are provided, along with concept expressions for minimum, maximum, value equality and ranges. Linear inequations are not provided, nor are named concrete objects, although these will be introduced in a later version of the schema. Ranges can be asserted for attributes using assertions like `rangeint` or `rangestring`. If no range is supplied, attributes have integers as their range by default.

An overview of the concrete forms of the operators is given in Figure 7.

7 Tell Syntax

A TELL request must contain in its body a `tells` element, which itself consists of a number of tell statements. An overview of the concrete forms of the operators is given in Figure 8. TELL requests are monotonic – i.e. once information has been told to a knowledge base, it can never be retracted or removed. The only such option available

Primitive Concepts	<top/> <bottom/> <catom name="CN" />
Boolean Operators	<and>E1... En</and> <or>E1... En</or> <not>E</not>
Property Restrictions	<some>R E</some> <all>R E</all> <atmost num="n">R E</atmost> <atleast num="n">R E</atleast> <iset>I1... In</iset>
Concrete Domain Expressions	<defined>A</defined> <stringmin val="s">A</stringmin> <stringmax val="s">A</stringmax> <stringequals val="s">A</stringequals> <stringrange min="s" max="t">A</stringrange> <intmin val="i">A</intmin> <intmax val="i">A</intmax> <intequals val="i">A</intequals> <inrange min="i" max="j">A</inrange>
Role Expressions	<ratom name="CN" /> <feature name="CN" /> <inverse>R</inverse> <attribute name="CN" /> <chain>F1... FN A</chain>
Individuals	<individual name="CN" />

Figure 7: Concept Language

is to release the knowledge base (see Section 5) and then start again. A TELL request must be made in the context of a particular knowledge base (which is identified via an attribute of the enclosing tells element).

Primitive Concept Introduction	<defconcept name="CN" /> <defrole name="CN" /> <deffeature name="CN" /> <defattribute name="CN" /> <defindividual name="CN" />
Concept Axioms	<impliesc>C1 C2</impliesc> <equalc>C1 C2</equalc> <disjoint>C1... Cn</disjoint>
Role Axioms	<impliesr>R1 R2</impliesr> <equalr>R1 R2</equalr> <domain>R E</domain> <range>R E</range> <rangeint>R</rangeint> <rangestring>R</rangestring> <transitive>R</transitive> <functional>R</functional>
Individual Axioms	<instanceof>I C</instanceof> <related>I1 R I2</related> <value>I A V</value>

Figure 8: Tell Language

The order of tell statements is unimportant.

The response to a tell will be a response message containing either an ok element, signifying that the statements were received and interpreted correctly, or an error element which may include an optional error code, message and detailed explanation.

In addition, an ok message may contain warnings about the tells received. For

example, the FaCT reasoner will happily process knowledge bases where primitive concepts are used without being explicitly introduced. However, it may be useful to warn the user if this has happened, as this may indicate a spelling or typographical mistake.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<tells
  xmlns="http://dl.kr.org/dig/2003/02/lang"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://dl.kr.org/dig/2003/02/lang
http://dl-web.man.ac.uk/dig/2003/02/dig.xsd"
  uri="urn:uuid:abcdefgh-1234-1234-12345689ab">
  <defconcept name="driver" />
  <equalc>
    <catom name="driver" />
    <and>
      <catom name="person" />
      <some>
        <ratom name="drives" />
        <catom name="vehicle" />
      </some>
    </and>
  </equalc>
  <defconcept name="person" />
  <defconcept name="vehicle" />
  <defrole name="drives" />
</tells>
```

Figure 9: Sample Tells

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<response
  xmlns="http://dl.kr.org/dig/2003/02/lang"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://dl.kr.org/dig/2003/02/lang
http://dl-web.man.ac.uk/dig/2003/02/dig.xsd">
  <ok>
    <warning message="Undeclared Name" code="99">Class Cat used
but not declared</warning>
  </ok>
</response>
```

Figure 10: Sample Response

Figure 9 shows a sample tell element. This defines three classes, vehicle, person and driver, where driver is further defined as being precisely those persons who drive a vehicle. Figure 10 illustrates a sample response. Here the reasoner has received the tells, but is reporting the use of an undeclared class.

8 Ask Syntax

An ASK request must contain in its body an asks element, which itself consists of a number of ask statements. Each ask statement must have an attribute id which supplies a unique identifier for the query (within the context of the particular collection of queries). This allows the presentation of multiple queries in one request, which may in turn allow the reasoner to optimise the processing of these queries. Each asks element must also have an attribute that identifies the knowledge base that the queries are being posed against. The value of this attribute should be a URI which identifies a

KB within the reasoner. An overview of the concrete forms of the queries is given in Figure 11.

Primitive Concept Retrieval	<allConceptNames/> <allRoleNames/> <allIndividuals/>
Satisfiability	<satisfiable>C</satisfiable> <subsumes>C1 C2</subsumes> <disjoint>C1 C2</disjoint>
Concept Hierarchy	<parents>C</parents> <children>C</children> <ancestors>C</ancestors> <descendants>C</descendants/> <equivalents>C</equivalents>
Role Hierarchy	<rparents>R</rparents> <rchildren>R</rchildren> <rancestors>R</rancestors> <rdescendants>R</rdescendants/>
Individual Queries	<instances>C</instances> <types>I</types> <instance>I C</instance> <roleFillers>I R</roleFillers> <relatedIndividuals>R</relatedIndividuals> <toldValues>I A</toldValues>

Figure 11: Ask Language

```
<?xml version="1.0"?>
<asks
  xmlns="http://dl.kr.org/dig/2003/02/lang"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://dl.kr.org/dig/2003/02/lang"
  http://dl-web.man.ac.uk/dig/2003/02/dig.xsd"
  uri="urn:uuid:abcdefgh-1234-1234-12345689ab">
  <satisfiable id="q1">
    <catom name="Vehicle"/>
  </satisfiable>
  <descendants id="q2">
    <and>
      <catom name="person"/>
      <some>
        <ratom name="drives"/>
        <catom name="vehicle"/>
      </some>
    </and>
  </descendants>
  <types id="q3">
    <individual name="John Smith"></individual>
  </types>
</asks>
```

Figure 12: Sample Query

Figure 12 shows a sample query request. This contains three queries. The first asks about the satisfiability of the Vehicle concept, the second asks for all those concepts subsumed by the description given, i.e. all the drivers. The third query asks for the known types of the given individual.

8.1 Interface Granularity

One key issue with the specification of an interface is the granularity of the operations supplied. For example, a criticism of the CORBA-FaCT interface was that in order to

construct a concept hierarchy (a common task for applications), the client had to make many requests to the server. This was undesirable due to the overhead involved with communication.

This becomes less of an issue when multiple requests are permitted in a single communication. For example, in order to determine the classification hierarchy, a client need only make two requests: one to determine the concepts in the hierarchy (for example through the use of an `<rdescendants>` or `<allConceptNames>` request, and one to determine all the immediate children of those concepts. This information is then sufficient to recreate the hierarchy without further communication with the reasoner.

9 Response Syntax

The schema contains a description of the responses expected of the server to ASK requests. An overview of the concrete forms of the operators is given in Figure 13.

The response to an ASK request must contain in its body a `responses` element, which itself consists of a number of responses – one for each query in the ASK. Each particular response must have an attribute `id` which corresponds to the identifier of a submitted query.

Error	<code><error/></code>
Boolean	<code><true/></code> <code><false/></code>
Concept Set	<code><conceptSet></code> <code><synonyms>S11... S1N</synonyms></code> <code><synonyms>SN1... SNM</synonyms></code> <code></conceptSet></code>
Role Set	<code><roleSet></code> <code><synonyms>R11... R1N</synonyms></code> <code><synonyms>RN1... RNM</synonyms></code> <code></roleSet></code>
Individual Set	<code><individualSet>I1... IN</individualSet></code>
Individual Pair Set	<code><individualPairSet></code> <code><individualPair>I1 I2</individualPair></code> <code><individualPair>J1 J2</individualPair></code> <code></individualPairSet></code>
Values	<code><sval>s</sval></code> <code><ival>i</ival></code>

Figure 13: Response Language

Figure 14 shows a possible response to the queries above. The answer to the first query is a boolean while the second yields a collection of concepts, including two synonymous concepts. The third query yields an error as the reasoner does not know about the individual given. Note that the responses do not necessarily occur in the order of submission.

10 Acknowledgements

Ralf Möller and Peter Crowther contributed greatly to the discussion leading to the drafting of this specification. The author would also like to thank other members of the Description Logic Implementation Group including Ian Horrocks, Sergio Tessaris,

```

<?xml version="1.0"?>
<responses
  xmlns="http://dl.kr.org/dig/2003/02/lang"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://dl.kr.org/dig/2003/02/lang
  http://dl-web.man.ac.uk/dig/2003/02/dig.xsd">
  <error code="99" id="q3" message="No Such Individual">The individual
  named does not exist in the knowledge base.</error>
  <true id="q1"/>
  <conceptset id="q2">
    <synonyms>
      <catom name="bus driver"/>
      <catom name="passenger service vehicle operator"/>
    </synonyms>
    <synonyms>
      <catom name="car driver"/>
    </synonyms>
  </conceptset>
</responses>

```

Figure 14: Sample Response

Peter Patel-Schneider, Volker Haarslev and Heiner Stuckenschmidt for their contributions. This work was supported by OntoWeb (EU grant IST-2000-25056) and Wonderweb (EU grant IST-2001-33052).

References

- [1] F. Baader, H.-J. Heinsohn, B. Hollunder, J. Muller, B. Nebel, W. Nutt, and H.-J. Profitlich. Terminological knowledge representation: A proposal for a terminological logic. Technical Memo TM-90-04, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH (DFKI), 1991.
- [2] Franz Baader and Phillip Hanschke. A Scheme for Integrating Concrete Domains into Concept Languages. Technical Report RR-91-10, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH (DFKI), 1991.
- [3] S. K. Bechhofer, C. A. Goble, A. L. Rector, W. D. Solomon, and W. A. Nowlan. Terminologies and Terminology Servers for Information Environments. In *Eighth International Workshop on Software Technology and Engineering Practice – STEP97*, pages 484 – 497, London, UK, 1997. IEEE Computer Society.
- [4] Sean Bechhofer, Ian Horrocks, Carole A. Goble, and Robert Stevens. Oiled: a reason-able ontology editor for the semantic web. In *Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence*, volume 2174 of *LNAI*, pages 396–408, Vienna, September 19-21 2001. Springer-Verlag.
- [5] Sean Bechhofer, Ian Horrocks, Peter F. Patel-Schneider, and Sergio Tessaris. A Proposal for a Description Logic Interface. In Lambrix et al. [10].
- [6] Enrico Franconi and Gary Ng. The i.com Tool for Intelligent Conceptual Modelling. In *7th Intl. Workshop on Knowledge Representation meets Databases (KRDB'00)*, Berlin, Germany, August 2000.
- [7] J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and Berners-Lee T. Hypertext Transfer Protocol – HTTP/1.1. <http://www.w3.org/Protocols/rfc2616/rfc2616.html>.
- [8] Volker Haarslev and Ralf Möller. Description of the RACER System and its Applications. In *Proceedings of the International Workshop on Description Logics (DL-2001)*, Stanford, USA, August 2001.

- [9] I. Horrocks. FaCT and iFaCT. In Lambrix et al. [10], pages 133–135.
- [10] P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, and P. Patel-Schneider, editors. *Proceedings of the International Workshop on Description Logics (DL'99)*, 1999.
- [11] Peter F. Patel-Schneider and Bill Swartout. Description logic specification from the KRSS effort, 1993.
- [12] World Wide Web Consortium. XML Schema. <http://www.w3.org/XML/Schema>.

A Error Codes

Standard error codes are provided to represent basic errors that may be raised by a DIG reasoner. In addition, particular reasoners may have specific error codes relating to the implementation. These should use codes in the range 900-999.

Class	Code	Explanation
General Errors	100	General Unspecified Error
	101	Unknown Request
	102	Malformed Request (XML error)
	103	Unsupported Operation
KB Errors	201	Cannot create new knowledge base
	202	Malformed KB URI
	203	Unknown or stale KB URI
	204	KB Release Error
	205	Missing URI
Tell Errors	300	General Tell Error
	301	Unsupported Tell Operation
	302	Unknown Tell Operation
Ask Errors	400	General Ask Error
	401	Unsupported Ask Operation
	402	Unknown Ask Operation

Table 1: Error Codes

B Semantics

This section presents a semantics for the language based on a DL semantics as in [1, 2]. An interpretation \mathcal{I} consists of a set $\Delta^{\mathcal{I}}$ – the domain – and an interpretation function $\cdot^{\mathcal{I}}$ that maps each concept name CN to a subset of the domain, each role name RN to a set-valued function (or collection of pairs of domain objects), and each feature name FN to a single-valued partial function. The current language specification supports only integers and strings as concrete domains. Each attribute name AN is thus mapped to a single valued function with range the integers (\mathcal{Z}) or strings (\mathcal{S}) and individual name IN to an element in the domain. We also assume the existence of an ordering on Strings, $\leq_{\mathcal{S}}$.

$$\begin{aligned} CN^{\mathcal{I}} &\subseteq \Delta^{\mathcal{I}} \\ RN^{\mathcal{I}} &: \Delta^{\mathcal{I}} \rightarrow 2^{\Delta^{\mathcal{I}}} \\ FN^{\mathcal{I}} &: \text{dom}(FN^{\mathcal{I}}) \rightarrow \Delta^{\mathcal{I}} \\ AN^{\mathcal{I}} &: \text{dom}(AN^{\mathcal{I}}) \rightarrow \mathcal{S} \cup \mathcal{Z} \\ IN^{\mathcal{I}} &\in \Delta^{\mathcal{I}} \end{aligned}$$

B.1 Expressions

Figure 15 shows the semantics of concept and role expressions (w.r.t. an interpretation \mathcal{I}). E represents an arbitrary concept expression, R a role expression, A an attribute or chain, I an individual expression, s a string value and i an integer.

B.2 Tells

Tells correspond to terminological axioms in the knowledge base. Figure 16 shows the semantics of the various tell operations. The `<defconcept>`, `<defrole>`, `<deffeature>`, `<defattribute>` and `<defindividual>` statements simply introduce new names.

B.3 Queries

The semantics of the query services are defined with respect to a knowledge base \mathcal{K} , and again follow the standard semantics defined in [1]. For satisfiability, subsumption, hierarchy traversal, instance retrieval and type requests, this is straightforward.

For some of the calls, however it is useful to provide some clarification of the intended semantics and expected responses.

For the `<allConceptNames/>` query, the reasoner should return all the known concept names. This should be returned as a `<conceptSet>` collection, i.e. the names should be grouped into sub collections of synonyms tagged using `<synonyms>`. Note also that for the sake of consistency, this collection should also include the `<top/>` and `<bottom/>` concepts.

In addition, whenever `<synonym>` sets are returned, the synonym set should include *all* synonyms. For example if an `<equivalents>` query is made with a concept atom as the argument, this atom should appear in the result set.

For the `<toldValues>` query, in the current version of the specification, this query is answered using pure retrieval, with no inference. The expected result of the query:

Concrete XML Form	Semantics
<top/>	$\Delta^{\mathcal{I}}$
<bottom/>	\emptyset
<catom name="CN"/>	$CN^{\mathcal{I}}$
<individual name="IN"/>	$IN^{\mathcal{I}}$
<and>E1...En</and>	$E1^{\mathcal{I}} \cap \dots \cap En^{\mathcal{I}}$
<or>E1...En</or>	$E1^{\mathcal{I}} \cup \dots \cup En^{\mathcal{I}}$
<not>E</not>	$\Delta^{\mathcal{I}} \setminus E^{\mathcal{I}}$
<some>R E</some>	$\{d \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(d) \cap E^{\mathcal{I}} \neq \emptyset\}$
<all>R E</all>	$\{d \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(d) \subseteq E^{\mathcal{I}}\}$
<atmost num="n">R E</atmost>	$\{d \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(d) \cap E^{\mathcal{I}} \leq n\}$
<atleast num="n">R E</atleast>	$\{d \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(d) \cap E^{\mathcal{I}} \geq n\}$
<iset>I1...In</iset>	$\{I1^{\mathcal{I}}, \dots, In^{\mathcal{I}}\}$
<defined>A</defined>	$\{d \in \Delta^{\mathcal{I}} \mid d \in \text{dom}(A^{\mathcal{I}})\}$
<stringmin val="s">A</stringmin>	$\{d \in \Delta^{\mathcal{I}} \mid s \leq_s A^{\mathcal{I}}(d)\}$
<stringmax val="s">A</stringmax>	$\{d \in \Delta^{\mathcal{I}} \mid A^{\mathcal{I}}(d) \leq_s s\}$
<stringequals val="s">A</stringequals>	$\{d \in \Delta^{\mathcal{I}} \mid A^{\mathcal{I}}(d) =_s s\}$
<stringrange min="s" max="t">A</stringrange>	$\{d \in \Delta^{\mathcal{I}} \mid s \leq_s A^{\mathcal{I}}(d) \leq_s t\}$
<intmin val="i">A</intmin>	$\{d \in \Delta^{\mathcal{I}} \mid i \leq A^{\mathcal{I}}(d)\}$
<intmax val="i">A</intmax>	$\{d \in \Delta^{\mathcal{I}} \mid A^{\mathcal{I}}(d) \leq i\}$
<intequals val="i">A</intequals>	$\{d \in \Delta^{\mathcal{I}} \mid A^{\mathcal{I}}(d) = i\}$
<inrange min="i" max="j">A</inrange>	$\{d \in \Delta^{\mathcal{I}} \mid i \leq A^{\mathcal{I}}(d) \leq j\}$
<ratom name="RN"/>	$RN^{\mathcal{I}}$
<feature name="FN"/>	$FN^{\mathcal{I}}$
<attribute name="AN"/>	$AN^{\mathcal{I}}$
<inverse>R</inverse>	$\{(d, d') \mid (d', d) \in R^{\mathcal{I}}\}$
<chain>F1...Fn A</chain>	$F1^{\mathcal{I}} \circ \dots \circ Fn^{\mathcal{I}} \circ A^{\mathcal{I}}$

Figure 15: Concept, Role and Attribute Expression Semantics

Concrete XML Form	Semantics
<code><impliesc>C1 C2</impliesc></code>	$C1^{\mathcal{I}} \subseteq C2^{\mathcal{I}}$
<code><equalc>C1 C2</equalc></code>	$C1^{\mathcal{I}} = C2^{\mathcal{I}}$
<code><impliesr>R1 R2</impliesr></code>	$R1^{\mathcal{I}} \subseteq R2^{\mathcal{I}}$
<code><equalr>R1 R2</equalr></code>	$R1^{\mathcal{I}} = R2^{\mathcal{I}}$
<code><disjoint>C1...Cn</disjoint></code>	$C1^{\mathcal{I}} \cap Cn^{\mathcal{I}} = \emptyset, i \neq j$
<code><domain>R C</domain></code>	$dom(R^{\mathcal{I}}) \subseteq C^{\mathcal{I}}$
<code><range>R C</range></code>	$ran(R^{\mathcal{I}}) \subseteq C^{\mathcal{I}}$
<code><rangeint>A</rangeint></code>	$ran(A^{\mathcal{I}}) \subseteq \mathcal{Z}$
<code><rangestring>A</rangestring></code>	$ran(A^{\mathcal{I}}) \subseteq \mathcal{S}$
<code><transitive>R</transitive></code>	$R^{\mathcal{I}} \circ R^{\mathcal{I}} \subseteq R^{\mathcal{I}}$
<code><functional>R</functional></code>	$\forall d \in dom(R^{\mathcal{I}}). R^{\mathcal{I}}(d) = 1$
<code><instanceof>I C</instanceof></code>	$I^{\mathcal{I}} \in C^{\mathcal{I}}$
<code><related>I1 R I2</related></code>	$I2^{\mathcal{I}} \in R^{\mathcal{I}}(I1^{\mathcal{I}})$
<code><toldValues>I A V</toldValues></code>	$I^{\mathcal{I}} \in dom(A^{\mathcal{I}}) \wedge A^{\mathcal{I}}(I^{\mathcal{I}}) = V^{\mathcal{I}}$
<code><sval>s</sval></code>	s
<code><ival>i</ival></code>	i

Figure 16: Tell Semantics

`<toldValues>I A</toldValues>`

is precisely the concrete value V for which there is a corresponding tell statement

`<toldValues>I A V</toldValues>`

in the knowledge base. For example, if the tells given to the reasoner include the statement:

```
<instanceof>
  <individual name="fred" />
  <rangeint min="20000" max="20000">
    <attribute name="salary" />
  </rangeint>
</instanceof>
```

the answer to the query:

```
<toldValues>
  <individual name="fred" />
  <attribute name="salary" />
</toldValues>
```

will **not** return the value 20000 (even though we might expect to be able to infer that this is the value for the given attribute).