A Proposal for a Description Logic Interface

Sean Bechhofer[†], **Ian Horrocks**[†], **Peter F. Patel-Schneider**[‡] and **Sergio Tessaris**[†]

[†]University of Manchester
Manchester, UK
seanb|horrocks|tessaris@cs.man.ac.uk

[‡]Bell Labs Research
Murray Hill, New Jersey, USA
pfps@research.bell-labs.com

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 the KRSS description logic specification [8].

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.

In such an environment, a DL might naturally be viewed as a self contained component, the details of whose implementation, and even the precise location in which its code is being executed, is hidden from the application [2]. 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.

2 A CORBA Server for DL Systems

We have used the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) [6] to build a generic DL server, to be used initially with both the FaCT and iFaCT systems [5]. CORBA was chosen because it is not tied to any particular language or platform. In particular, CORBA can be used with both Lisp and Java running on both Unix and Microsoft platforms.

The CORBA solution has all the advantages mentioned above.

- It facilitates the use of the Lisp implementations by non-Lisp client applications, for example in the Tambis (Transparent Access to Multiple Biological Information Systems) project, where the DL server is used by a Java client [1].
- The generic API is defined using CORBA's Interface Definition Language (IDL), which can be mapped to various target languages.
- The application communicates with the DL via a CORBA Object Request Broker (ORB). The DL server and client application may or may not be running on the same physical machine.
- It would be possible to substitute FaCT or iFaCT with another DL reasoner, for example DLP [7], without client applications even being aware of the change.

It has been decided *not* to pass concepts and roles as objects: treating them as objects does not seem natural (as they have no functionality), and could lead to a significant increase in overheads (as determining their structure might require many object requests via the ORB). However, the CORBA IDL does not support the definition of the kinds of recursive data type that would be required for the representation of DL concepts and roles.

The solution adopted is to pass concepts and roles as single data items using eXtended Markup Language (XML) [9]. The advantages of using XML are that it is becoming a widely accepted standard, it naturally lends itself to the definition or recursive structures, and there are parsers available for several languages (including Lisp and Java). The disadvantage of XML is that it is more verbose and (arguably) less human readable than the familiar Lisp style syntax. However, it must be emphasised that XML is NOT intended as a user interface medium, but only for data exchange between components of the server and between the server and client applications. Figure 1 shows an example of a definition in both KRSS and XML syntax of the concept "Proud-Parent", a person whose children are all either Doctors or Lawyers.

KRSS:

```
(and Person
     (some child Person)
     (all child (or Doctor Lawyer)))
```

XML:

```
<CONCEPT>
  <AND>
    <PRIMITIVE NAME="Person"/>
    <SOME>
      <PRIMROLE NAME="child"/>
      <PRIMITIVE NAME="Person"/>
    </SOME>
    <ALL>
      <PRIMROLE NAME="child"/>
      <OR>
        <PRIMITIVE NAME="Doctor"/>
        <PRIMITIVE NAME="Lawyer"/>
      </OR>
    </ALL>
  </AND>
</CONCEPT>
```

Figure 1: KRSS and XML syntax

3 System Architecture

Only a minimal interface to the DL reasoner has been defined. It is intended that additional functionality and more sophisticated interfaces be provided by other components, which would be clients of the DL reasoner. Client applications would then interact with an interface component. All these interactions make use of the ORB bus, as shown in Figure 2.

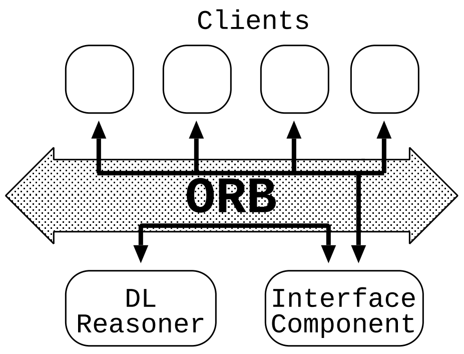

Figure 2: FaCT server Architecture

This architecture provides a mechanism for developing a complete DL system with interchangeable reasoning and interface components. It is even envisaged that sub-components of the DL system, such as subsumption reasoner, Abox reasoner and hierarchy maintenance, could be separated. This would facilitate the cooperative development of systems and the rapid integration of new components, regardless of their implementation language.

4 The Client Interface Component

The interface provided by the DL reasoner is little more than ORB access to a Lisp evaluate and print loop. A separate Interface Component has been implemented (in Java), and provides a more sophisticated object oriented API for use by client applications. This API is seen as an object in the CORBA namespace, and provides operations which clients use to interact with the DL; the interface seen by clients is thus separated from the real reasoning engine. Even this API is very simple, and it is anticipated that many applications would want to augment it either directly or by interposing another level of indirection. Moreover, the current API only considers Tbox reasoning, and would need to be extended if an Abox reasoner were added to the system.

The interface conforms to a standard “tell and ask” format: facts are asserted to the knowledge base (KB) and queries answered without the user specifying when or how reasoning should be performed. In order to improve efficiency, and to support the (future) possibility of multi-user access to a KB, the interface has a simple transaction control mechanism. This mechanism could also be augmented with partial (complete) roll-back: the ability to undo the last (an arbitrary number of) transactions.

Before performing any tell operations, a client must perform a begin_transaction operation; if this is successful it can be followed by any number of tell operations. A transaction can be ended either with an end_transaction or an abort_transaction operation, the latter having the effect of discarding all the tell operations performed since the transaction began. Any ask operations performed during a transaction will be answered in the normal way, but will not reflect any of the tell operations in the incomplete transaction. As well as providing a simple locking mechanism, grouping tell operations in this way gives the system a hint as to when it might be sensible to perform some reasoning, without introducing an explicit “classify” operation.

Errors are signaled by raising exceptions (a standard feature of CORBA). The different types of exception are:

kr_transaction_required The requested operation can only be performed in the context of a transaction.

kr_op_unimplemented The requested operation is not implemented by the server.

kr_expr_error Concept or role syntax error. This also covers the case of unimplemented operators.

The small number of exception types is due to the simplicity of the interface and the decision not to consider any kind of KB condition (e.g., concept or KB unsatisfiability) as an error. Many kr_op_unimplemented errors could of course be raised, depending on the capabilities of the DL reasoner.

Return	Operation	Parameters	Meaning
void	defconcept	CN	CN $\sqsubseteq \top$
void	defrole	RN	RN $\sqsubseteq \top \times \top$
void	implies_c	C_1, C_2	$C_1 \sqsubseteq C_2$
void	equal_c	C_1, C_2	$C_1 \doteq C_2$
void	implies_r	R_1, R_2	$R_1 \sqsubseteq R_2$
void	equal_r	R_1, R_2	$R_1 \doteq R_2$
void	transitive	RN	RN is transitive
void	functional	RN	RN is functional
void	clear		$\mathcal{T} := \emptyset$

Table 1: Tell operations

The available tell and ask operations are summarised in Table 1 and Table 2, where CN is a concept name, RN is a role name, C is a concept, R is a role, $\mathcal{CN}$ is a set of sets of concept names, $\mathcal{RN}$ is a set of sets of role names, $\mathcal{P}$ is a triple ($\mathcal{CN}_1$,$\mathcal{CN}_2$,$\mathcal{CN}_3$), and $\mathcal{T}$ is the set of axioms that make up the KB. The $\mathcal{CN}$ and $\mathcal{RN}$ data types are used to return sets of named concepts or roles, each of which may have a set of synonyms; in such cases no one name can or should be preferred over the others. The $\mathcal{P}$ data type is used to return a concept's position in the hierarchy, were it to be classified, in terms of its direct subsumers ($\mathcal{CN}_1$), synonyms ($\mathcal{CN}_2$) and direct subsumees ($\mathcal{CN}_3$).

All concept and role names are assumed to be atomic primitives, and the defconcept and defrole operations are provided only for completeness. For efficiency, some optimisation would be required (either in the interface component or the DL reasoner), e.g., the conversion of general axioms to definition axioms whenever possible [4].

5 Discussion

It will be noted that the ask interface supports only taxonomic and logical queries; there is no provision for the retrieval of concept definitions or other facts directly asserted to the KB. This is consistent with the view of the DL as a reasoning component and with the specification of the tell interface. The storage and retrieval of asserted facts is not a logical operation, and if it is required this functionality could be provided within client applications. On the other hand, given that the KB consists of an arbitrary set of asserted facts, a name P may have no "definition" (an axiom with P as its left hand side), or it may have many. Moreover, the set of concept expressions that subsume or are equivalent to P may depend on other non-definitional axioms in the KB.

We are aware of the effort in the KR community to develop a common API for accessing conceptually diverse KRSs, in particular the promising OKBC project by the Knowledge Systems Lab (KSL) at Stanford University and the AI centre at SRI International [3]. OKBC is intended to define a common API for different KRSs, with the goal of enabling the sharing and reusability of KBs written by knowledge engineers using different KRSs.

From a DL perspective, the problem we envisaged in adopting OKBC is that, although it provides some limited support for logical KBs, its underlying assumption is of a frame based knowledge model. Forcing a DL KRS to conform to the OKBC view seems unnatural, and it was therefore decided that the KRSS specification was a more sensible starting point. However, we may conceive of an OKBC compliant API as a wrapper around our API, where the OKBC services are implemented as logical services.

References

[1] P. G. Baker, A. Brass, S. Bechhofer, C. Goble, N. Paton, and R. Stevens. Tambis: Transparent access to multiple bioinformatics information sources: an overview. In *Proceedings of the Sixth International Conference on Intelligent Systems for Molecular Biology, ISMB98*, 1998.

[2] 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.

[3] V. K. Chaudhri, A. Farquhar, R. Fikes, P. D. Karp, and J. P. Rice. OKBC: A programmatic foundation for knowledge base interoperability. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and of the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pages 600–607. AAAI Press, jul 1998.

[4] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics.* PhD thesis, University of Manchester, 1997.

[5] I. Horrocks. FaCT and iFaCT. In *Collected Papers from the International Description Logics Workshop (DL'99)*, to appear.

[6] The Object Management Group. *The Common Object Request Broker: Architecture and Specification*, 1998.

[7] P. F. Patel-Schneider. DLP system description. In E. Franconi, G. De Giacomo, R. M. MacGregor, W. Nutt, C. A. Welty, and F. Sebastiani, editors, *Collected Papers from the International Description*

Return	Operation	Parameters	Meaning
$\mathcal{CN}$	direct_supers_c	CN	direct subsumers of CN
$\mathcal{CN}$	all_supers_c	CN	all subsumers of CN
$\mathcal{CN}$	direct_subs_c	CN	direct subsumees of CN
$\mathcal{CN}$	all_subs_c	CN	all subsumees of CN
$\mathcal{RN}$	direct_supers_r	RN	direct subsumers of RN
$\mathcal{RN}$	all_supers_r	RN	all subsumers of RN
$\mathcal{RN}$	direct_subs_r	RN	direct subsumees of RN
$\mathcal{RN}$	all_subs_r	RN	all subsumees of RN
$\mathcal{P}$	taxonomy_position	C	taxonomy position of C
boolean	satisfiable	C	$(\bot \sqsubset C)$?
boolean	subsumes	C_1, C_2	$(C_1 \sqsubseteq C_2)$?
boolean	equivalent	C_1, C_2	$(C_1 \doteq C_2)$?

Table 2: Ask operations

Logics Workshop (DL'98), pages 87–89. CEUR, May 1998.

[8] P. F. Patel-Schneider and B. Swartout. Description logic specification from the KRSS effort, June 1993.

[9] Extensible markup language (XML) 1.0. W3C Recommendation TR REC-xml-19980210, February 1998. Editors T. Bray, J. Paoli, C. M. Sperberg-McQueen.