Enabling Semantic Web Programming by Integrating RDF and Common Lisp

Ora Lassila
Nokia Research Center, 5 Wayside Road, Burlington, Massachusetts, USA

Abstract: This paper introduces "Wilbur", an RDF and DAML toolkit implemented in Common Lisp. Wilbur exposes the RDF data model as a frame-based representation system; an object-oriented view of frames is adopted, and RDF data is integrated with the host language by addressing issues of input/output, data structure compatibility, and error signaling. Through seamless integration we have achieved a programming system well suited for building "Semantic Web" applications.

1. Introduction

Common Lisp [24] is a programming language that has enjoyed great popularity in the AI community. Despite its somewhat waning use, it can still be considered one of the most expressive mainstream programming languages. Because of its somewhat unique integration of rich data structures with the language itself, Common Lisp offers the interesting possibility of integrating RDF [18, 19] and DAML [10, 11] data with a programming language, therefore making it easier to build software that takes advantage of the "Semantic Web" [3].

This paper will discuss "Wilbur", a Common Lisp -based open source toolkit for RDF and DAML. Wilbur includes an API (Application Programming Interface) which allows the underlying RDF data to be treated as a frame system, essentially providing an object-oriented view of the data. The relationship between frame-based representation, object-oriented modeling, and RDF is straightforward [20], but an even more interesting aspect is the synergistic potential of integrating a programming language with a frame system [15]. Many frame systems have offered some type of programming support such as access-oriented behavior [e.g., 13, pp.30-32] or some other type of "slot daemons" (for example, both CRL and KEE allowed a Lisp function to be invoked when certain operations were being performed on a slot). Tight integration, however, would in practice have to involve not only integration of the frame system's and the programming language's type systems, but also leveraging the programming language's native programming model and facilities (such as method invocation).

2. RDF Toolkits

RDF data consists of nodes and attached attribute/value pairs. Nodes can be any Web resources, including other RDF nodes. Attributes are named properties of nodes, and their values are either atomic (text strings) or other nodes. The essence of RDF is this model of nodes, properties and their values. In addition to the node-centric view the RDF model can be seen as

directed, labeled graphs (DLGs). The nodes are the vertices of a graph, and the properties name the edges. Therefore, if X has a property Y with the value Z, we can think of X and Z linked by an edge labeled Y, pointing from X to Z.

To make construction of "RDF-savvy" software easier, a number of RDF toolkits have recently appeared, offering functionality that goes beyond mere parsing. Examples of these toolkits are Redland [2], Jena [21], and the ICS-FORTH RDFSuite [1]. These toolkits are typically implemented in either Java or C/C++.

"Wilbur" is Nokia Research Center's open source toolkit for RDF and DAML, written in Common Lisp. Like other RDF toolkits, it offers an API for manipulating RDF data (graphs, nodes, etc.) as well as parsing functionality (parsers not only for XML-encoded RDF and DAML but also for "plain" XML [5] since one written in Common Lisp did not exist when the Wilbur project was started[1]; it also offers a simple HTTP client API for accessing remote URLs for the same reason). Wilbur also offers a frame system API on top of the RDF data API, including a simple query language. Wilbur strives for tight integration of RDF data with the intrinsic features of Common Lisp.

Generally, Wilbur implements the RDF data model by providing four abstract interfaces (and their concrete implementations):

1. The class `node` represents nodes of an RDF graph. Each node may have a URI (Universal Resource Identifier) string associated with it, in which case we consider the node to be *named*; nodes without a URI are called *anonymous* (the reader is referred to the discussion of URIs and their printed representation below).
2. A mapping from URI strings to nodes is provided by the class `dictionary`. The system uses a single default dictionary where all named nodes are placed. The unique mapping from URI strings to `node` instances allows us to implement strict *read/print correspondence* for nodes (described below).
3. The class `triple` represents labeled arcs of an RDF graph. A triple consists of a *subject* (a `node` instance), a *predicate* (also a `node` instance), and an *object* (either a `node` instance or a string, although in the current implementation any Common Lisp object can be used); each triple also has an associated *source* (also a `node` instance), designating the file or HTTP URL from which the triple was originally parsed.
4. Collections of triples are stored in databases (instances of class `db`). The upper level API of the system assumes a single default database, but also exposes a lower-level API where the database can be specified explicitly (allowing software to be constructed which makes use of multiple databases). Simple query functionality is provided for se-

---

[1] Wilbur's XML parser (written in Common Lisp) has an interface similar to SAX 1 [22]. The parser was written with RDF's needs in mind and does not, for example, support DTDs (except for entity declarations).

lecting triples from a database, similar to the "find" interface of the Stanford RDF API [23] (not to be confused with the Wilbur frame query language described later).

For debugging purposes, the object inspector of the Macintosh Common Lisp was extended to allow easy browsing of RDF graphs.[2]

3. Integration Issues

Our two previous frame systems, BEEF [12, 16] and PORK [17], both addressed the issue of integrating object-oriented programming with frame-based representation. BEEF (which predated practical implementations of the Common Lisp Object System) added object-oriented programming features to a frame system, whereas PORK approached the issue from the opposite direction by taking an object-oriented programming language and adding features of frame-based representation to it; PORK used the Common Lisp *metaobject protocol* [14] to extend the Common Lisp Object System (CLOS).

Wilbur, as a frame system API overlaid on RDF, takes a lower-level approach to integration, by allowing manipulation of RDF graphs. Future development may still address programming issues taking either the "BEEF-approach" (adding programming features to a frame system) or the "PORK-approach" (adding frame features to a programming language). In Wilbur, the RDF/CLOS integration focuses on the following areas:

- ease of use of Common Lisp data structures with RDF,
- issues of input and output of RDF data in a "Common Lisp -friendly" manner, and
- the use of the Common Lisp *condition mechanism* for signaling unexpected situations.

3.1. Reading and Printing RDF Data

To be able to use RDF data seamlessly in an interactive Common Lisp environment, this data must have a printed representation which can be read back into a Common Lisp system. Common Lisp defines this quality, known as *read/print correspondence* [24, p.509], as follows:

> "Ideally, one could print a LISP object and then read the printed representation back in, and so obtain the same identical object. In practice this is difficult and for some purposes not even desirable. Instead, reading a printed representation produces an object that is (with obscure technical exceptions) `equal`[3] to the originally printed object."

The former approach is called "strict read/print correspondence" and the latter "non-strict"; many Common Lisp data structures (such as lists and strings) are non-strict, whereas some (such as symbols) are strict. Wilbur provides *strict* read/print correspondence for nodes.

---

[2] Similar to BBN's DAML Viewer [7]

[3] `equal` is a Common Lisp predicate for structural similarity.

URIs are used internally throughout Wilbur: they give unique identity to nodes. In order to avoid having to write (and read) full URIs, which typically are rather long, the system provides an abbreviated syntax, based on the idea of namespace-qualified names in XML [4]. For example, if we introduce a mapping for the prefix "foo" as follows:

```
"foo" → "http://foo.com/schema#"
```

then we have

```
"foo:bar" → "http://foo.com/schema#bar"
```

Although the XML namespace specification does not specifically define concatenating the expanded form of the prefix with the name part, Wilbur adopts the RDF convention of turning each qualified name into a single (concatenated) URI string.

Wilbur uses the Common Lisp *read macro* mechanism to incorporate the expansion of abbreviated URIs into the *reader* (i.e., the Common Lisp parser). Any expression of the form `!foo:bar` is turned into an instance of Wilbur's `node` class and placed into a dictionary which maps URI strings to `node` instances. This allows references to nodes to be embedded in Common Lisp source files, thus enabling one to embed RDF Data in compiled (binary) files. Wilbur uses the notion of a "forward reference" to a node in cases where the abbreviated URI could not be resolved. When a missing prefix-to-URI mapping is introduced, the system updates the affected nodes by resolving the URIs. This approach is similar to the forward reference model of PORK which allowed one to easily construct circular data structures without having to worry about the order in which named objects were introduced [17].

For printing data structures, Common Lisp defines [24, p.510] that

> "When `print` produces a printed representation, it must choose arbitrarily from among many printed representations. It attempts to choose one that is readable."

The `print-object` method for the Wilbur `node` class uses any existing prefix-to-URI mapping to determine a possible abbreviated form of a node's URI, and subsequently produces a printed representation which can be read in if necessary.

The Wilbur toolkit has two separate parsers, one conforming to the RDF Model and Syntax specification [19] and another conforming to the DAML+OIL reference description [11]. The RDF parser supports all features[4] of the specification, including reification of complete descriptions, reification of individual statements, and the attribute namespace ambiguity. The parser is "near-streaming" and is internally based on a state machine where SAX-like parsing events serve as transition inputs.

---

[4] Except "rdf:aboutEachPrefix" which probably no-one supports.

The DAML parser (class `daml-parser`) is implemented as an extension of the RDF parser (i.e., as a subclass of `rdf-parser`) and adds support for the DAML collection syntax specified using `rdf:parseType="daml:collection"`.

3.2. Integrating Data Structures

The Wilbur frame API itself is quite simple, basically offering functions for creating frames, for adding values to a slot, for deleting values from a slot, and for reading a slot's values. Frames in Wilbur form graphs when slot values are other frames. Wilbur introduces a query language for selecting subgraphs from these graphs (in other words selecting sets of nodes from RDF graphs). Query expressions are patterns expressed as regular expressions with arc labels (slots, i.e., RDF properties) as atoms, using the following operators and "pseudo-labels":

- **Sequence**: the operator `:seq` matches a sequence of n steps in the graph, consisting of subexpressions e_1,e_2,...,e_n; the operator `:seq+` is similar except any sequence e_1,e_2,...,e_k for k in [1...n] will match.
- **Disjunction**: the operator `:or` matches any one of n subexpressions e_1,e_2,...,e_n.
- **Repetition**: the operator `:rep*` matches the transitive closure of subexpression e; the operator `:rep+` is the same as `(:seq e (:rep* e))`.
- **Inverse**: satisfaction of `(:inv e)` requires the path defined by the subexpression e to be matched in reverse direction.
- **Container membership**: the atom `:members` will match any of the `rdf:_1`, `rdf:_2`, `rdf:_3`, etc. container membership properties.
- **Wildcard**: the atom `:any` will match any label.

The Wilbur query language is similar to the BEEF path grammar [16] which, in turn, was a simplification of the CRL path grammar [8, 9]. Given a "root" node (i.e., a search start point) and a path (a query expression), Wilbur provides functions for retrieving either the first reachable node or all reachable nodes, and for determining whether a path exists between two specified nodes. These functions make it easy to turn RDF graphs into Common Lisp list structures. For example, given a DAML collection (constructed as a "dotted-pair" list using the properties `daml:first` and `daml:rest`), the following query expression will turn it into a Common Lisp list:

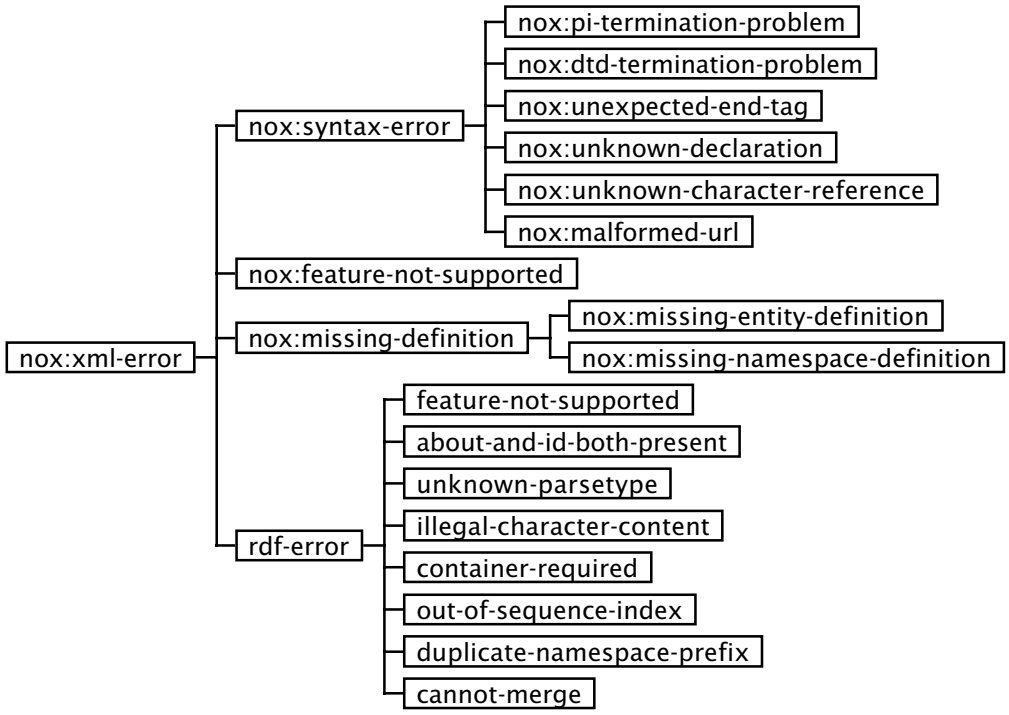
```
(:seq (:rep* !daml:rest) !daml:first)
```

As mentioned before, the Wilbur DAML parser supports the DAML collection syntax and correctly generates dotted-pair lists.

3.3. Dealing with Unexpected Situations

The Common Lisp *condition system* is a powerful mechanism for raising signals when unexpected situations are encountered. When a condition is signaled, instead of reporting an error,

the calling program may choose to catch the signal and allow the execution to continue on from the point where the signal was raised (or caught). Wilbur defines a rich taxonomy of classes for various types of unexpected conditions, and takes full advantage of the condition system's ability to "ignore" errors. The following figure illustrates this taxonomy (note that condition classes in the "nox" package are generated by the XML parser):



As a general rule, all errors of the XML parser are signaled as "non-continuable" (i.e., they abort parsing) whereas all errors of the RDF and DAML parsers are signaled as "continuable" (using the Common Lisp function `cerror`) and allow parsing to continue if the user or the calling program so chooses. The rich taxonomy allows fine-grained mapping of errors to remedial behaviors.

4. Future Work

Several additional features of the toolkit are currently at an experimental stage. These include an RDF *serializer*, capable of producing textual XML from triple databases, and a *schema validator*, capable of checking triple database consistency against the constraints defined by the RDF Schema specification [6].

Both the serializer and the validator make extensive use of the query language. For example, in order to find out whether a slot value (here denoted by *x*) satisfies the (disjunctive) range constraints of a property (here denoted by *p*), the following query can be executed:

```
(relatedp x
          '(:seq !rdf:type
                 (:rep* !rdfs:subClassOf)
                 (:inv !rdfs:range)
                 (:rep* (:inv !rdfs:subPropertyOf)))
          p)
```

Note that the call `(relatedp A B C)` determines whether node C can be reached from node A via path B.

In addition to RDF 1.0 and DAML+OIL, Wilbur will have "plug-in" parsers for the "RDF-like" DMoz Open Directory format and for the alternate RDF syntax "N3".

Other future work will focus on DAML and supporting requirements of the DAML community (for example, we are working on an OKBC interface to the Wilbur frame system), as well as supporting changes introduced by the W3C RDF Core Working Group for the next version of RDF.

5. Conclusions

The Wilbur toolkit attempts to create a programming environment for RDF and DAML by closely integrating some of the representational features with the programming features provided by Common Lisp and CLOS. Issues in integrating input and output of RDF data are addressed, as well as compatibility of RDF and Common Lisp data structures. A query language is introduced to make it easier to select parts of RDF graphs and convert them to Common Lisp data structures.

Exposing RDF as a frame system and allowing programmers to use the full power of Common Lisp makes it easier to create "Semantic Web" applications. Using the frame paradigm also makes it easier to understand RDF (and data models expressed using RDF).

References

[1] Sofia Alexaki, Vassilis Christophides, Gregory Karvounarakis, Dimitris Plexousakis, and Karsten Tolle: "The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases", in: S.Staab et al (eds.): "Proceedings of the Second International Workshop on the Semantic Web", May 2001

[2] David Beckett: "The Design and Implementation of the Redland RDF Application Framework", in: Proceedings of the Tenth International World Wide Web Conference, WWW10, May 2001

[3] Tim Berners-Lee, James Hendler, and Ora Lassila: "The Semantic Web", Scientific American, May 2001

[4] Tim Bray, Dave Hollander, and Andrew Layman: "Namespaces in XML", W3C Recommendation, World Wide Web Consortium, January 1999

[5] Tim Bray, Jean Paoli, C.M.Sperberg-McQueen, and Eve Maler: "Extensible Markup Language (XML) 1.0 (Second Edition)", W3C Recommendation, World Wide Web Consortium, October 2000

[6] Dan Brickley & R.V.Guha: "Resource Description Framework (RDF) Schema Specification 1.0", W3C Candidate Recommendation, World Wide Web Consortium, March 2000

[7] Mike Dean & Kelly Barber: "DAML Viewer", www.daml.org/viewer/

[8] Mark S. Fox: “Knowledge Representation for Decision Support”, in: L.B.Methlie & R.H.Sprague (eds.): “Knowledge Representation for Decision Support Systems”, Elsevier, 1985

[9] Mark S. Fox, J.Wright, and D.Adam: “Experiences with SRL: An analysis of a frame-based knowledge representation”, in: Expert Database Systems, Benjamin/Cummings, 1985

[10] James Hendler & Deborah L. McGuinness: “DARPA Agent Markup Language”, IEEE Intelligent Systems 15(6):72-73

[11] Frank van Harmelen, Peter F. Patel-Schneider and Ian Horrocks (eds.): "Reference description of the DAML+OIL (March 2001) ontology markup language", working document of the DARPA Agent Markup Language program, March 2001

[12] Juha Hynynen & Ora Lassila: “On the Use of Object-Oriented Paradigm in a Distributed Problem Solver”, AI Communications 2(3):142-151, 1989

[13] Peter D. Karp: “The design space of frame knowledge representation systems”, Technical Report 520, SRI International AI Center, 1992

[14] Gregor Kiczales, Jim des Rivières, and Daniel G. Bobrow: “The Art of the Metaobject Protocol”, MIT Press, 1991

[15] Ora Lassila: “Frames or Objects, or Both?”, Workshop Notes from the Eight National Conference on Artificial Intelligence (AAAI-90): Object-Oriented Programming in AI, American Association for Artificial Intelligence, July 1990 (also Report HTKK-TKO-B67, Department of Computer Science, Helsinki University of Technology, 1990)

[16] Ora Lassila: “BEEF Reference Manual - A Programmer's Guide to the BEEF Frame System”, Second Version, Report HTKK-TKO-C46, Department of Computer Science, Helsinki University of Technology, 1991

[17] Ora Lassila: "PORK Object System Programmer's Guide", Report CMU-RI-TR-95-12, The Robotics Institute, Carnegie Mellon University, 1995

[18] Ora Lassila: “Web Metadata: A Matter of Semantics”, IEEE Internet Computing 2(4):30-37

[19] Ora Lassila & Ralph R. Sw ick: "Resource Description Framework (RDF) Model and Syntax Specification", W3C Recommendation, World Wide Web Consortium, February 1999

[20] Ora Lassila & Deborah L. McGuinness: "The Role of Frame-Based Representation on the Semantic Web", Report KSL-01-02, Knowledge Systems Laboratory, Stanford University, 2001

[21] Brian McBride: “Jena: Implementing the RDF Model and Syntax Specification”, in: Steffen Staab et al (eds.): “Proceedings of the Second International Workshop on the Semantic Web - SemWeb'2001”, May 2001

[22] David Megginson: “SAX 1.0: The Simple API for XML”, www.megginson.com/SAX/SAX1/

[23] Sergey Melnik: "RDF API Draft", working document, Stanford University, 1999

[24] Guy L. Steele, Jr: “Common Lisp - the Language, 2nd ed.”, Digital Press, 1990

Acknowledgements

The author would like to thank the following individuals for their advice during the Wilbur project and during the preparation of this article: Jessica Jenkins, Marcia Lassila and Louis Theran, as well as the three anonymous reviewers whose suggestions proved invaluable.

Although portable to any Common Lisp platform, the Wilbur toolkit was developed entirely using Digitool’s “Macintosh Common Lisp” (which the author considers to be a fantastic software development environment).

Wilbur is an open source software project. More information about the project is available at http://purl.org/NET/wilbur/.