Common Lisp Support for Semantic Web Programming

Ora Lassila

Research Fellow

Agent Technology Group, Nokia Research Center

October 2003

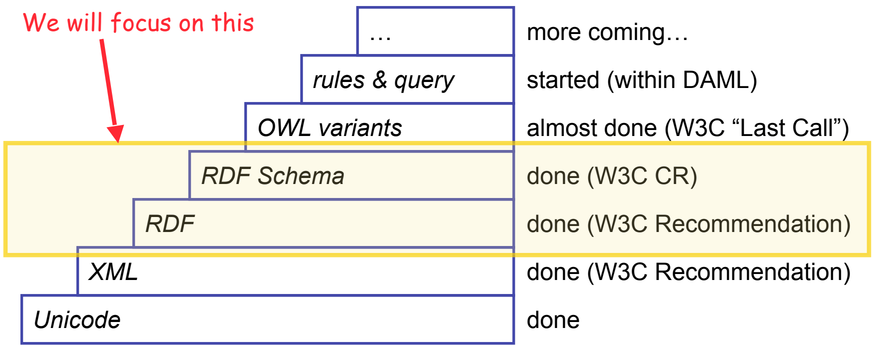
Talk Outline

- Semantic Web overview
- Simple frame-based programming
- "Wilbur" toolkit
- Experiments
- Q & A

Semantic Web

- **WWW content is “machine-readable” but not “machine-understandable”**
  - consequently, automating anything is hard…
- **Semantic Web is understood as an attempt to make WWW content friendlier for machines and automated systems to process**
- **Rough approach: apply KR on a WWW-wide scale**
  - logic- and frame-based formalisms
- **Standardization of some aspects underway (at W3C)**

Stepping Towards the Semantic Web



We will focus on this

…	more coming…
rules & query	started (within DAML)
OWL variants	almost done (W3C “Last Call”)
RDF Schema	done (W3C CR)
RDF	done (W3C Recommendation)
XML	done (W3C Recommendation)
Unicode	done

- **Semantic Web is built in a layered manner**
- **Not everybody needs all the layers**

About RDF

- **Directed labeled graphs**
- **Two “views”**
  - Graph view (“semantic networks”): semi-structured graph data
  - Object-oriented view: objects with properties
- **Nodes named with URIs**
  - bnodes (“anonymous”) nodes have no URI
  - arcs labeled with nodes
- **Graphs decompose into *object - attribute - value* tuples**
  - in RDF parlance: “subject - predicate - object”
- **Our approach to programming (with) this**
  - think of RDF nodes as “frames”, and arcs as “slots”

Common Lisp & Frame Systems

- **Traditional approaches to “procedural attachment”**
  - object-oriented approach
  - access-oriented approach, slot dæmons
- **Our previous approaches**
  - BEEF: frame system with added OOP features
    - SCAM, a lightweight derivative of BEEF, flew with NASA’s Deep Space 1
  - PORK: OOP language with added frame system features
    - using CLOS metaobject protocol
- **Wilbur approach: low-level**
  - input/output
  - easy translation of RDF data structures to CL data structures
  - exception handling
- **Wilbur exposes a “frame API” where RDF graphs look like frames, slots, and values**

Input & Output

- **Printed representation ←→ input of "parsed" data**
  - CL "read/print equivalence"
    - strict: same object identity
    - non-strict: "similar" objects
  - Node (URI) literals, embedding in source files
    - XML NS -style qualified names

Example: `!rdf:type` → a node instance with URI
`"http://www.w3.org/1999/02/22-rdf-syntax-ns#type"`

- **Parsers**
  - RDF/XML (w/ DAML extensions)
  - N-Triples
  - "dialects" (e.g., DMoz)
- **HTTP Client**

Data Structure Conversions

- **Collections**
  - RDF collection model is "awkward"…
  - Common Lisp collection model is natural and well integrated into the language
- **Query language**
  - easy selection of nodes from an RDF graph
  - pattern matching of subgraphs
    - query patterns are regular expressions
    - queries define traversal through a graph
  - Easy conversion of collections to CL lists

Example: query pattern

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

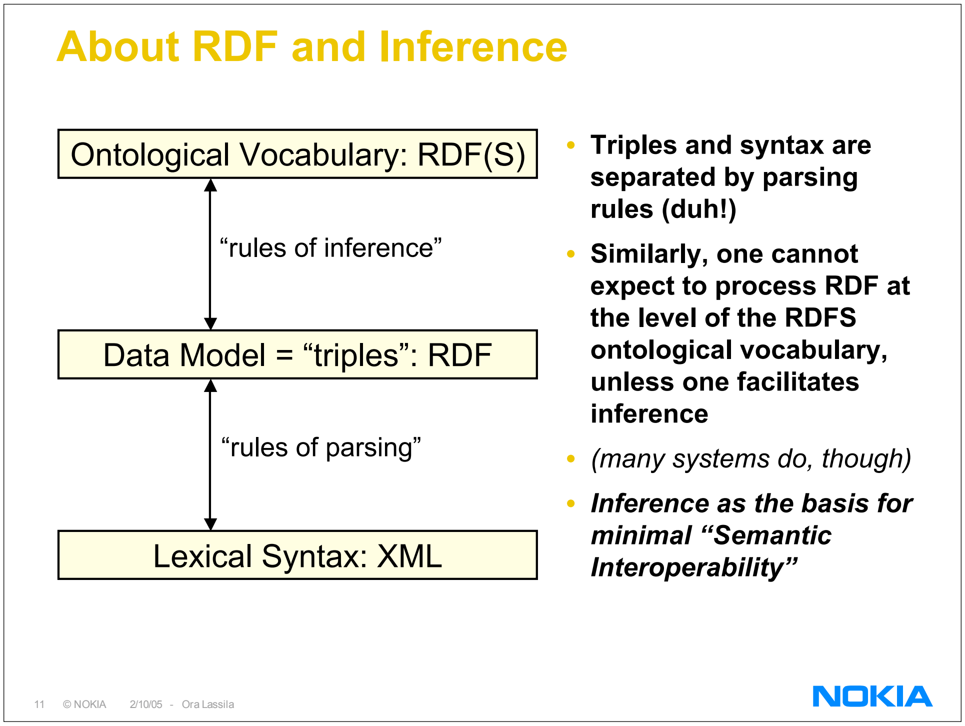
converts an RDF list into a CL list

Exception Handling

- **CLOS has a very powerful exception signaling mechanism**
- **Wilbur signals all errors and anomalies**
  - RDF conditions are always signaled as “continuable”
- **Fine-grained condition hierarchy allows selective response by calling program**

Wilbur as an Experimentation Platform

- **Standards tracking**
  - RDF Core WG
  - WebOnt WG
  - DAML-S
- **“Hiding” inference from the application programmer**
- **Reflecting the RDF type system onto CLOS**



About RDF and Inference

Ontological Vocabulary: RDF(S)

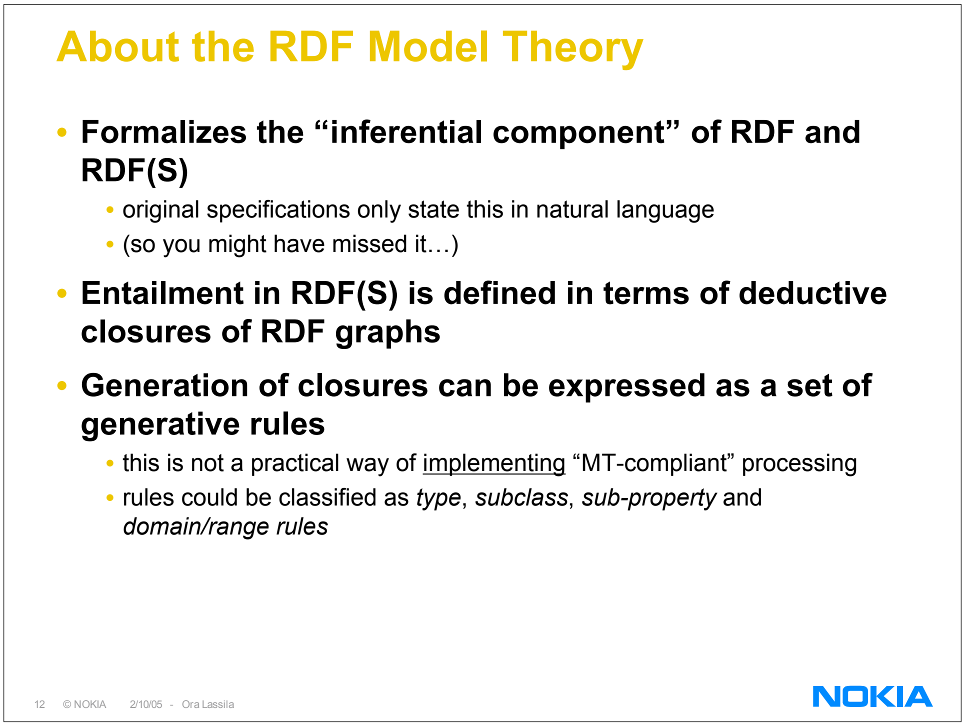
"rules of inference"

Data Model = "triples": RDF

"rules of parsing"

Lexical Syntax: XML

- **Triples and syntax are separated by parsing rules (duh!)**
- **Similarly, one cannot expect to process RDF at the level of the RDFS ontological vocabulary, unless one facilitates inference**
- *(many systems do, though)*
- ***Inference as the basis for minimal "Semantic Interoperability"***



About the RDF Model Theory

- **Formalizes the "inferential component" of RDF and RDF(S)**
  - original specifications only state this in natural language
  - (so you might have missed it…)
- **Entailment in RDF(S) is defined in terms of deductive closures of RDF graphs**
- **Generation of closures can be expressed as a set of generative rules**
  - this is not a practical way of <u>implementing</u> "MT-compliant" processing
  - rules could be classified as *type*, *subclass*, *sub-property* and *domain/range rules*

Options for Closure Processing

- **Option A:**
  1. generate new triples
  2. query against the “new” graph
- **Option B:**
  1. query against the original graph, but
  2. add triples on demand (“on the fly”)
- **Other options exist:**
  - combinations of A & B + various caching schemes
  - optimizations of forward-chaining rule processing
  - backward chaining
  - etc.
- **Tradeoffs: processing time vs. memory consumption**

Closure Generation

- **Wilbur has a simple access API:**

  $$value \in A_{lookup}(frame, slot) \Leftrightarrow \langle frame, slot, value \rangle \in D$$

  where *D* is the database of “triples”

- **We would like to enhance the API as follows:**

  $$value \in A(frame, slot) \Leftrightarrow \langle frame, value \rangle \in IEXT(I(slot))$$

  where *IEXT(p)* is the binary relational extension of a property and *I(x)* is the RDF(S) interpretation relation of the RDF model theory

- **Simplistically, the solution is to “rewrite” *slot* expressions:**

  $$A(frame, slot) = A_{lookup}(frame, slot')$$

  where *slot’* is the rewritten (possibly complex) access path

Closure Generation (contd.)

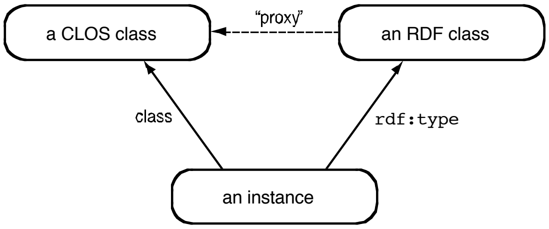
- **Delay computation**
  - compute *on demand* (even at the expense of CPU time spent)
  - split computation between insertions and queries
- **Some RDF “features” are more common than others**
  - use of subclasses is common
  - sub-properties are used, but are less common
  - sub-properties of `rdfs:subPropertyOf` are rare
- **Practical prototype**
  - implemented on top of Wilbur (66 lines of code in CLOS)
  - dynamically maintains the closure while the graph is modified
- **Notion of “semantic interoperability” in RDF hinges on getting the inference right**
  - must not leave this to application programmers (analogies to parsing)
  - few reasons to use RDF otherwise

Replicating RDF Type System in CLOS

- **Typically, the RDF type system is implemented in another (object-oriented) language**
  - implementation language features (method invocation, class definition) not available for RDF
  - external RDF model accessed via some API
  - standard approach for all Java-based RDF toolkits
- **What if you could reflect the RDF type system onto the implementation language?**
  - RDF classes would become classes in the implementation language
  - all implementation language features would be available in RDF
  - requires good reflective capabilities from the implementation language
- **CLOS is a good choice for this experiment…**

Bootstrapping the RDF Class Graph

- **“Proxy principle”**
  - reader macro defined for easy reference to proxy classes
    - RDF class: !foo:Bar
    - proxy: ?foo:Bar



a CLOS class

“proxy”

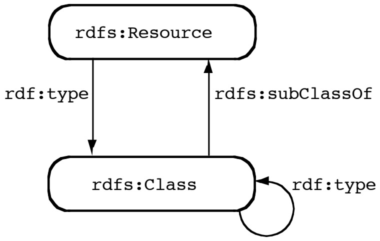
an RDF class

class

rdf:type

an instance

- **Minimal RDF class graph will be constructed**
  - all other (RDF) classes are created automatically



rdfs:Resource

rdf:type

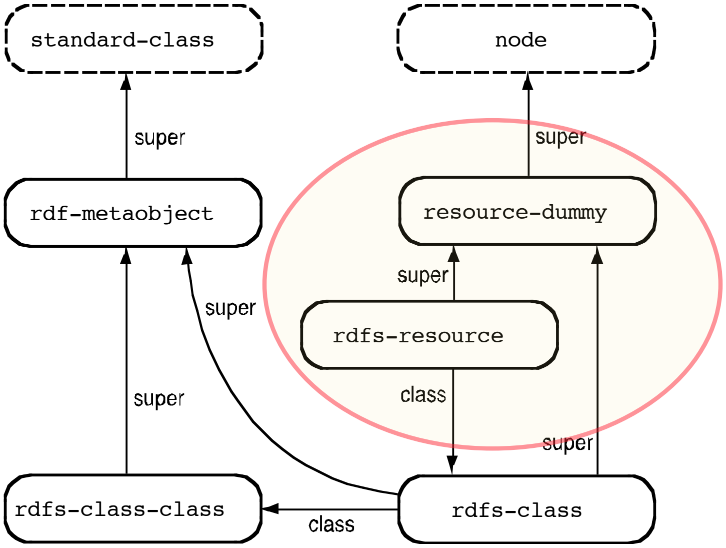
rdfs:subClassOf

rdfs:Class

rdf:type

NOKIA

CLOS Class Graph: initial version



standard-class

node

super

super

rdf-metaobject

resource-dummy

super

super

rdfs-resource

super

class

super

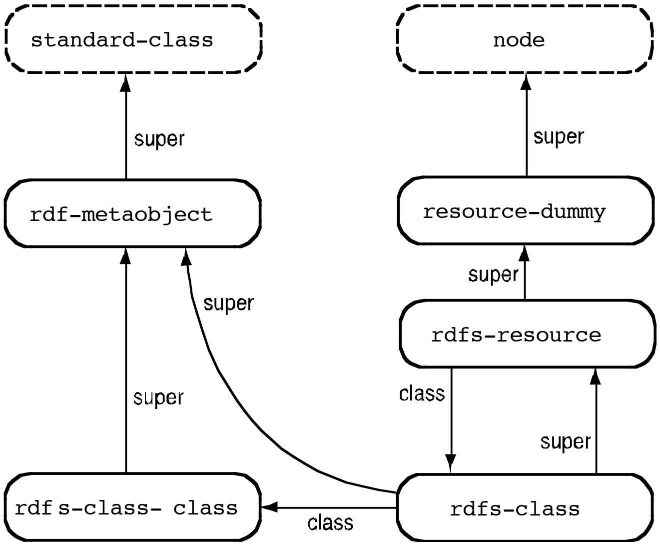
rdfs-class-class

class

rdfs-class

NOKIA

CLOS Class Graph: after tweaking



Slot Access

- **CLOS has single-valued slots**
- **RDF can have “repeated” properties**
  - these could be seen as multiple-valued slots
- **We need a “transactional” API for slot values**
  - mimic PORK’s slot access protocol
- **MOP makes things easy**

Simple Examples

- **Handling RDF containers**

```
(defmethod as-list ((self ?rdfs:Container))
  (wilbur:all-values self :members))
(defmethod as-list ((self ?rdf:List))
  (wilbur:all-values self
                     `(:seq (:rep* !rdf:rest)
                            !rdf:first)))
```

- **OWL special properties**

```
(defmethod slot-value-using-class
      ((class rdf-metaobject)
       (instance ?rdfs:Resource)
       (slot ?owl:TransitiveProperty))
  (wilbur:all-values instance `(:rep+ ,slot)))
```

Conclusions

- **An attempt at a programming environment for RDF**
  - Did we succeed? Still too early to tell…
  - (we are looking for more good users)
- **Input and output integration makes it easy to interface with RDF data**
  - reflecting the type system makes things even easier
  - frame-based view makes it easy to understand RDF
- **Query language makes it easy to “convert” RDF data structures to Common Lisp**
- **Support for inference is critical**

Questions?

- http://purl.org/net/wilbur/
- mailto:ora.lassila@nokia.com

Reaction from test audience…