The Use of Lisp in Semantic Web Applications

Yarden Katz
University of Maryland
MIND Lab
8400 Baltimore Ave.
College Park MD 20742, USA
yarden@umd.edu

James Hendler
University of Maryland
Computer Science Department
College Park MD 20742, USA
hendler@cs.umd.edu

ABSTRACT

The Semantic Web adds a layer of logic and metadata to the current World Wide Web. By utilizing traditional Artifical Intelligence (AI) and Knowledge Representation (KR) techniques for both the construction of new documents and linking of existing ones, the Semantic Web facilitates machine-to-machine (or "agent-to-agent") communication. Lisp's proven reliability and flexibility in AI and KR make it ideal for constructing intelligent Semantic Web applications. In this paper, we survey the current use of Lisp on the Semantic Web, and suggest some potential uses of it in the future. We conclude the paper with descriptions of Lisp in selected Semantic Web projects that demonstrate its strength and usability.

General Terms

Semantic Web, AI, Lisp, Web Services, Reasoning, Planning

1. INTRODUCTION: LISP DOMINATES AI

Lisp's highly dynamic nature and flexible handling of data has established it as the obvious tool choice for many types of complex Artificial Intelligence applications. Additionally, some of the most exciting research in the fields of Description Logics and other Knowledge Representation areas has used Lisp as its vehicle of representation.

Despite its success in those fields, Lisp has not gained widespread popularity in the realm of Web programming. The current arena of web development is dominated primarily by languages such as Perl and Python. Java also plays a significant role in web client/server software.

The popularity of those languages in this domain is legitimate: they are highly portable, fairly easy to learn and most importantly, they provide extensive text manipulation functionality which is one of the most needed features in current web-related applications that process non-semantic documents (HTML, TXT, TEX, etc.) Moreover, these languages came into popularity at the same time as the Web, and thus their architecture and supporting libraries/environments were evolving to be more compelling for web-oriented usage. Lisp has reached widespread popularity in programming circles before the Web's existence and therefore followed a different path of development. Although it has been shown that Lisp's unique handling of data structures is equally suitable to process even those non-semantic formats[22], Lisp has never caught on as a Web language.

As the current Web evolves into the Semantic Web, application developers will require a different set of functionality from their tools. Easy access to text manipulation (e.g., Perl's built-in regular expressions) will no longer be the prime focus of every Web application. Reasoning, planning and formal representation of data will become ubiquitous, standard tasks that need to be performed by agents to fully utilize the new semantic data on the web. Lisp is already capable of providing such functionality as shown from its extensive history in those fields.

We believe that Lisp can, in the Semantic Web application domain, take on the role of both the "behind the scenes engine" (serving as a reasoner or planner, for example) as well as the application interface, or front end. In the little use that it has received in web projects, Lisp conventionally only served the former role, leaving the latter to other languages such as Perl, Python and Java.

2. SEMANTIC WEB LANGUAGES

The World Wide Web Consortium (W3C) is overseeing developments of several Semantic Web languages.

The Semantic Web relies on XML[10] as the basis of its representation syntax. The Resource Description Framework (RDF)[36] is an XML application designed to allow statements in the form of *subject*, *predicate*, *object* (a "triple") to be made about resources. RDF Schema (RDFS) in turn adds classes and class/property relationships (such as *domain* and *range*) to RDF[18].

The ontology layer consisted of DAML+OIL[34], now succeeded by the Web Ontology Language (OWL)[33], a W3C candidate recommendation. Three species of OWL are available: Lite, DL and Full.

OWL builds on the capbilities of RDFS in several ways[16]:

International Lisp Conference 2003 NYC, New York, USA

- Enhanced restrictions: the ability to restrict a property to *all values* or *some* values (one or more) of a class.
- Enhanced class relations: set operations (*unionOf*, *intersection*, *complementOf*, *disjointWith*) on classes.
- Enhanced property relations: inverse, functional, transitive, and symmetric properties.
- Arbitrary cardinality for properties.

While OWL Lite and OWL DL provide expressive logic functionality, OWL Full's capabilities extend beyond the boundaries of conventional Description Logics.

2.1 Building on XML

XML allows information on the web to be modelled structurally. To represent that "John" is the *name* of an *employee*, we can write:

```
<Employee>
  <Name>John</Name>
</Employee>
```

Using a DTD, we can also express that the `Name` tag must only appear within an `Employee` tag. Additionally, XML data types can be used to enforce that only string literals can be valid `Name` tag values. However, all of these are structural properties; nothing about the *meaning* or *semantics* of the concepts employee and name was conveyed. We have simply asserted that in our document, there exists some tag (`Employee`) whose child tag (`Name`) has the string literal value "John."

The Resource Description Framework (RDF) builds on XML to describe resources. A resource can be anything: a person named John, the concept of an employee, John's workplace, etc. Resources are represented by URIs. To enhance our previous example, we can add that "John works for Google.com" using RDF:

```
<Employee
rdf:about="http://john-homepage.com/John>
 <name
 rdf:datatype="http://www.w3.org/2001/XMLSchema#st-
ring">
 John
 </name>
 <worksFor rdf:resource="http://www.google.com"/>
</Employee>
```

The above snippet states that the resource `http://john-homepage.com/John`, has the `name` "John," and a `worksFor` property whose value is the resource `http://www.google.com`. We have also restricted the values of the `name` property to XML strings.

Using ontology languages, we can further enhance our model by adding semantics. For example, we can define the concepts "employee" and "name" as follows: an employee is a person, and a name can only be given to people. Similarly, it may also be useful to express that people only work for people[1]. To accomplish this, we use OWL:

```
<owl:Class rdf:ID="Employee">
  <rdfs:subClassOf
   rdf:resource="http://john-homepage.com/Person"/>
</owl:Class>
<owl:ObjectProperty
 rdf:ID="http://john-homepage.com/name">
  <rdfs:domain
   rdf:resource="http://john-homepage.com/Person"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="Person">
  <owl:Restriction>
    <owl:onProperty
     rdf:resource="http://john-homepage.com/worksFor"/>
    <owl:allValuesFrom
     rdf:resource="http://john-homepage.com/Person"/>
  </owl:Restriction>
</owl:Class>
```

We now know that employees are a subset of people, a name can only be given to people[2], and a person can only work for another person.

Given this new layer of semantics in our data, there are several entailments that we can make. Very simply, since an employee is a person, and John is an employee, John is also a person. By the same logic, the class Employee can also be used as a value for the `worksFor` property due to the same subsumption relation.

We can now refer to the above collection of definitions (an *ontology*) in our RDF. In practice, it is wise to use definitions from popular, prewritten ontologies (about people or employees, in our case) rather than reinvent our own.

3. LISP ON THE WWW

While Lisp has not reached the mainstream in the world of Web programming, several robust Web tools in Lisp are available. Some of the well-known tools are described herein.

On the server-side, the Common Lisp Hypermedia Server[26] (CL-HTTP) stands out as a stable and full-featured alternative to the average webserver. CL-HTTP served the White House Publication System, both as a web and email interface, focusing on fault tolerance (handling failed mail processing) and allowing complex querying of the document base.

Franz Inc. has also released an open-source webserver, AllegroServe[13] (AServe), that was modified by the Lisp community to run on multiple Lisp implementations[1]. Our use of AServe in a Semantic Web application is described in a later section.

[1]It is important to note that in practical ontology modelling such strong restrictions should be carefully evaluated.
[2]Note our valid use of an RDF Schema property, `domain`, in an OWL document. Integration with RDF Schema was an important goal in the design of OWL.

Finally, Araneida[4] provides a similar webserver environment for the SBCL implementation[5]. A popular Araneida-powered application is CLiki[2], the Lisp community's wiki pages.

To construct a usable client-side, several web development toolkits are available. onShore Development's IMHO[29] (Internet Metahumble Objects) provides extensive functionality for handling web sessions and serving pages via a sophisticated template system. It runs alongside the Apache webserver. IMHO's interface is consistently object-based, allowing customization of methods for specific types of content. WebCheckout[30] is an example of a popular and sophisticated institutional management system (capable of asset management, equipment scheduling, classroom-related tasks and more) developed using IMHO.

4. LISP IN CURRENT SEMANTIC WEB

Significant research and development in Lisp is already ongoing in the areas of planning, reasoning and querying of Semantic Web data. A brief overview of some of those applications is provided below.

4.1 Accessing Semantic Data with Wilbur

Wilbur[25] is Nokia's open-source toolkit for Semantic Web programming. Wilbur's object-oriented interface, implemented using CLOS, provides access to a DAML parser, validating RDF and XML parsers, triple store and frame-based RDF query language[24].

Wilbur's straight-forward API lets developers access RDF data in a "node-centric"[23] manner. A "database" (or triple store) is automatically instantiated upon parsing of a document, into which all triples are asserted as CLOS objects (this allows for interesting extensability, described in a later section.) By default, Wilbur's initial database contains a static set of triples along with an RDF Schema that define and enforce RDF's basic properties and classes.

4.1.1 *Ivanhoe: Frame-based path language for RDF*

Ivanhoe is Wilbur's path language for accessing RDF. Consistent with Wilbur's node-centric view of RDF, it provides frame-like access to RDF graphs. Properties (or arcs of a graph) are represented as slots while subjects and objects are either nodes or fillers (actual values-in practice these are always a string.) Querying is handled by the path grammer described below.

Ivanhoe's path grammer is implemented as a set of regular expressions that allow for complex querying of RDF documents. The basic building blocks of the grammer are shown in Table 1.

For example, to find all the parents of a given class, the following query is used:

```
(all-values !dcms:UndergraduateStudent
            '(:rep+ !rdfs:subClassOf))
→
(!dcms:Student !dcms:Swapper)
```

Similarly, we can test whether a given class is related to another via subclassing:

Table 1: Ivanhoe Path Language operators

Operator	Description
`seq`	Sequence
`rep+`	Repetition (0 or more)
`rep*`	Repetition (1 or more)
`inv`	Inverse
`or`	Disjunction
`value/all-values`	Single path value/all values to path

```
(relatedp !dcms:UndergraduateStudent
          '(:rep+ !rdfs:subClassOf) !dcms:Swapper)
→
T
```

In the above example, although the class `UndergraduateStudent` is not a direct subclass of `Swapper`, the path language expression still matches because of the use of the eager `:rep+` operator.

Ivanhoe's path language operators allow for basic, axiomatic inferencing to be built into RDF applications. As shown above, it is trivial, for example, to trace all of the `subClassOf` pointers in a given document to locate the root superclass. While this approach may result in incomplete sets of answers for certain knowledge bases, it is sufficient for many Semantic Web applications that work with a defined set of data. If it is reasonable to expect your knowledge base to consistenly have all `subClassOf` relationships made explicit (*any class P that subsumes class Q has Q* `subClassOf` *P*), then the power and overhead of a full-fledged reasoner is unnecessary for simple queries such as *What is the topmost superclass of P* or *What is the most specific subclass of Q*. Ivanhoe's path language is sufficient for such ubiquitous and minimal inference requirements.

5. REASONING ABOUT ONTOLOGIES

Reasoning is one of the most powerful and ubiquitous features of Semantic Web applications. In order to increase interoperability of different agents, inferences must be made about the data being exchanged. Lisp's accomplishments and capabilities in formal Description Logic reasoning extend to the Semantic Web.[3]

5.1 RACER: Renamed ABox and Concept Expression Reasoner

RACER[15] is a robust Description Logic reasoning engine geared toward Semantic Web applications. It is capable of answering queries with sensitivity to DAML+OIL ontologies.

RACER can be used as the reasoning service for the reasonable ontology editor OilEd[7], not natively, but via the DIG interface[6].

5.2 FaCT: Fast Classification of Terminologies

FaCT[17] is a highly optimized Description Logic reasoner. It currently serves as the reasoning engine behind OilEd.

[3]Many commonly used Description Logics are far more expressive than Semantic Web languages such as OWL or DAML+OIL.

FaCT's Common Lisp source code is freely available, although efforts are already underway to rewrite FaCT in C++. Like RACER, FaCT is able to answer queries via the DIG interface as well as in a client/server setup.

6. MIND SWAP'S LISP DEVELOPMENTS

MIND SWAP (Semantic Web Agents Projects) is a Semantic Web research group based in the Maryland Information and Network Dynamics (MIND) lab at the University of Maryland, College Park. MIND SWAP is headed by Professor James Hendler. We have created several Semantic Web tools, including: an ontology editor[19], a multimedia markup environment[21], a web services composition tool[31], a Semantic Web mail client[35], and most recently, a tableaux-based OWL reasoner[32].

Several ongoing MIND SWAP projects in the areas of web services, reasoning, planning and web development are written in Lisp. Our projects utilize some of the existing Semantic Web Lisp libraries mentioned previously, as well as other popular open-source Lisp libraries and toolkits. An overview of the projects and their implementation is provided below.

6.1 Planning for Web Services

Planning is a crucial requirement for interoperability of web services. As services become more widespread, it becomes harder for the end-user to determine what set of services are required to complete the desired task. Additionally, manually matching the inputs and outputs of one service to another can often be a tedious process that is better left to automated software.

6.1.1 The Scientific American Article Scenario

The canonical use case for planning on the Semantic Web is described in Berners-Lee, Hendler and Lassila[8].

The scenario is a simple everyday task that is shared among two people: Lucy and Pete are equally partitioning the task of chauffeuring their mother to a series of doctor appointments. The siblings' agents must come up with an optimal plan such that the assigned driving times for each sibling do not conflict with their previously scheduled events, and that their mother is able to attend the doctor's scheduled appointments. In this scenario we make the assumption that Semantic Web agents will have a web service interface to other entities in the scenario. More specifically, the doctor's office's agent will be able to provide patient-related information about the *available appointment times* and *prescribed treatment*, while Lucy's agent will be able to query Pete's agent for information (such as *Pete's availability* from his schedule) and vice versa. Finally, *pharmacy locating* and related drug services are also used.

An implementation of a solution to this problem must return a plan detailing what and how web services should be called, in order. Our current implementation is powered by SHOP2[28], an award winning AI planner written in Lisp.

6.1.2 Implementation: DAML-S to SHOP2

SHOP2 (Simple Hierarchical Ordered Planner) is a generalized HTN planning system. SHOP2 works by Ordered Task Decomposition; this makes it suitable for our scenario, where the order of tasks in our planned solution is the same order in which they will later be executed as web services.

Our scenario's web services are in DAML-Services (DAML-S)[3], a high-level layer for describing the operations and inputs/outputs of a Semantic web service.[4] In order to integrate our web services with SHOP2, Allegro's jLinker[11] interface between Java and Lisp is utilized.

A set of algorithms written in Java[37] to convert DAML-S descriptions to to SHOP2 operators is applied to the web services. Once in this format, SHOP2 is able to reduce our planning problem to an ordered set of web services that must be called with their respective values.

Finally, an interface that lets the user specify what services must be called is also available.

6.1.3 A Planned Solution

Returning to Lucy and Pete's scenario, a sample plan based on the two agents' parameters is generated by SHOP2. The abstract, abbreviated plan is provided below. A much longer and more detailed plan is also generated.

```
(((!NS_ORDERPRESCRIPTIONCVS PRESCRIPTION1)
  (!MAKEAPPOINTMENTEMG1 APPOINTMENT78)
  (!UPDATESCHEDULELUCY APPOINTMENT78)
  (!MAKEAPPOINTMENTH1 APPOINTMENT182)
  (!UPDATESCHEDULELUCY APPOINTMENT182)
  (!MAKEAPPOINTMENTD APPOINTMENT203)
  (!UPDATESCHEDULEPETE APPOINTMENT203)))
```

Due to the use of the jLinker component, SHOP2 is unaware of URIs or web-specific data; all web service execution is handled by Java. Therefore, in the final plan, methods and operators cannot directly refer to the URIs of the web services that must be executed. Instead, our Java implementation assigns unique identifiers for every available planning possibility (multiple appointments times, prescriptions pickup times, locations of pharmacies.) For example, the above set of `APPOINTMENT` identifiers are encoded in RDF in the `AppointmentList` property:

```
<ns1:AppointmentList rdf:ID="AppointmentList65"
                     parseType="daml:collection">
   ...
   <ns1:Appointment rdf:ID="Appointment66">
      <ns1:day>20021216</ns1:day>
      <ns1:availability>yes</ns1:availability>
      <ns1:hour>9</ns1:hour>
   </ns1:Appointment>
   <ns1:Appointment rdf:ID="Appointment78">
      <ns1:day>20021219</ns1:day>
      <ns1:availability>yes</ns1:availability>
      <ns1:hour>9</ns1:hour>
  </ns1:Appointment>
  <ns1:Appointment rdf:ID="Appointment70">
```

[4]The lower-level details are abstracted to WSDL groundings, where the specific mechanical workings of the web service are specified.

```
      <ns1:day>20021217</ns1:day>
      <ns1:availability>yes</ns1:availability>
      <ns1:hour>9</ns1:hour>
  </ns1:Appointment>
  ...
</ns1:AppointmentList>
```

This mapping is saved to allow the Java component to look up the details of every appointment as it appears in the SHOP2 plan. The same is done with location preferences, pharmacy locators, available time slots for drivers, and other intricate details of the solution that are not included in the above abstract plan.

6.1.4 Future work

In the near future, we intend to create another version of the web services planning system that is written purely in Lisp, eliminating the use of jLinker. Wilbur and the upcoming native SOAP and WSDL interfaces[12] from Franz Inc. will allow Semantic Web resources to be manipulated directly from SHOP2, rather than relying on the Java components to do the web-oriented work.

6.2 Reasoning Services: OWLLisaKB

OWLLisaKB[20] is a rule-based reasoner for the OWL language. It is powered by the LISA[38] forward-chaining inference engine and the Wilbur RDF toolkit. OWLLisaKB's primary focus is on OWL Full, whose scope extends beyond the scope of Description Logics.

6.2.1 LISA

One of the primary advantages of the LISA system is its ability to reason with arbitrary CLOS objects. Since the Wilbur triple store uses CLOS objects to internally represent triples, very little translation is required to share data between the two systems. Figure 2 illustrates the interaction between Wilbur and LISA that occurs internally in OWLLisaKB.

6.2.2 Mapping Triples for Reasoning

The first step is to process the semantic data in Wilbur. This results in a database of CLOS objects, each representing a triple. In order for LISA to reason about these CLOS objects, they must be explicitly asserted into our knowledge base; this is the second step. Once asserted as facts, a series of synchronization rules (along with some intervention from Lisp[5]) keep the LISA knowledge base up to date with Wilbur's triple database and vice versa.

When the LISA inference engine runs, a series of rules and preasserted facts are applied to the newly asserted triples. Our rules express basic OWL and RDF semantics and our preasserted facts express simple RDF truths such as, *`type` is of type `Property`* and *`Class` is a subclass of `Resource`*.

The mapping that took place earlier between Wilbur triples and LISA facts allows us to match triple elements in rules

[5]We overloaded Wilbur's `add-triple` method to automatically assert every newly added triple as a fact in LISA. This mechanism is only triggered for the global Wilbur database so that separate triple databases that are unaffected by the Wilbur to LISA synchronization could still be created.

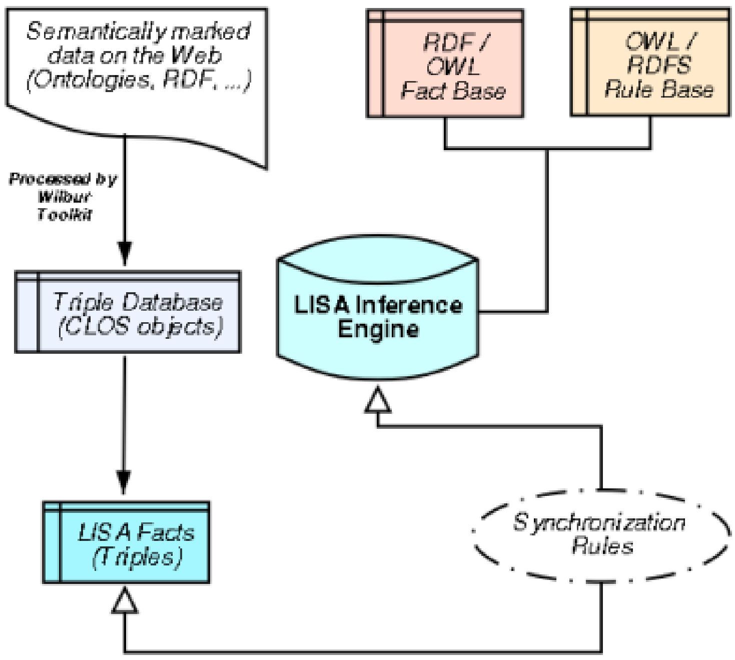

Semantically marked data on the Web (Ontologies, RDF, ...)
RDF / OWL Fact Base
OWL / RDFS Rule Base
Processed by Wilbur Toolkit
Triple Database (CLOS objects)
LISA Inference Engine
LISA Facts (Triples)
Synchronization Rules

Figure 1: LISA and Wilbur interaction in OWLLisaKB

as slot names. Consider the RDFS rule below for enforcing the subclass relationship in instances: *An instance I of a subclass C is an instance of the parent class P, provided that I is not of type P.*

```
(defrule subclass-instances ()
  (triple
   (subject ?C)
   (predicate
    "http://www.w3.org/2000/01/rdf-schema#subClassOf")
   (object ?P))
  (triple
   (subject ?I)
   (predicate
    "http://www.w3.org/1999/02/22-rdf-syntax-ns#type")
   (object ?C))
  (not
   (triple
    (subject ?I)
    (predicate
     "http://www.w3.org/1999/02/22-rdf-syntax-ns#type")
    (object ?P)))
  =>
  (assert
   ?I
   "http://www.w3.org/1999/02/22-rdf-syntax-ns#type"
   ?P))
```

The components of a Wilbur CLOS triple are now accessed through the slot name holders `subject`, `predicate`, `object`.

OWLLisaKB is currently being tested against the W3C OWL test cases[9]. Our focus has been mainly on PETs (Positive

Entailment Tests) but we are also experimenting with several techniques for passing the suggested consistency and inconsistency tests.

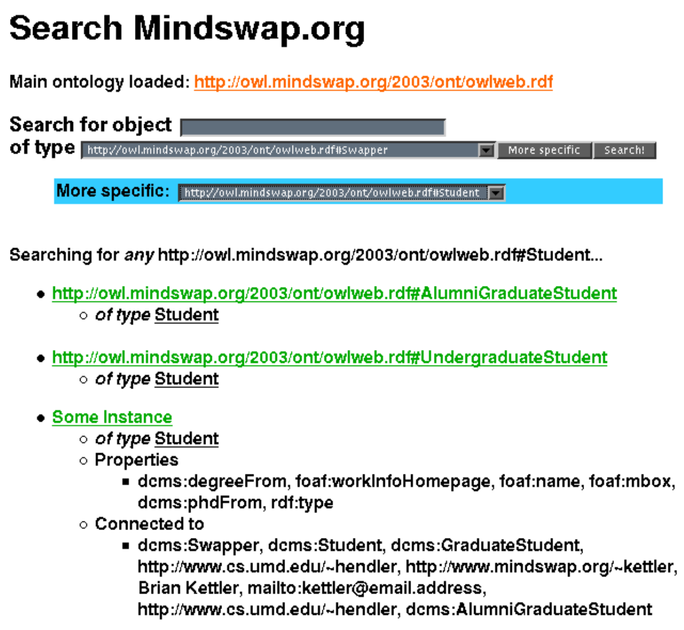
The simplicity of OWLLisaKB coupled with LISA's performance and efficiency make it attractive for use in Lisp applications that need native access to inference. An example of such application is the MIND SWAP Search page, described below. In addition, non-Lisp applications that prefer a lighter-weight component to call out to (be it via a web service interface, TCP socket, or FFIs) might find a rule-based reasoner to be more suitable than a full-fledged reasoning environment like RACER or FaCT.

6.3 MIND SWAP Search: A Pure Lisp Application

A prime example of a pure Lisp Semantic Web application is the MIND SWAP Search page. The search page allows arbitrary searches of both instances and concepts (ABox and TBox queries) in our OWL-backed website[27]. In order to get complete results, a reasoner must be used to fully capture the meaning and implications of our semantic data.

MIND SWAP Search relies on OWLLisaKB and Franz's AServe. OWLLisaKB performs all of the reasoning behind the search page. During initialization, OWLLisaKB reasons about the entire MIND SWAP database (a total of around 7,500 triples) in approximately two minutes. All search operations are then performed as a series of Ivanhoe path language queries, as illustrated earlier, to the updated Wilbur database that contains the OWLLisaKB inferred triples.

AServe provides the web interface (Figure 2) for MIND SWAP Search. As argued by Graham[14], subroutine-like functionality in web application greatly increases usability. AServe easily facilitates this functionality by allowing the developer to bind[6] blocks of code to specific URIs. This simulates to the user Graham's subroutine functionality; a user can follow a link, enter some data, and return to the main page where the updated data will appear, just as if the link were a function call that returned to the main body of the program. The search page uses this structure for displaying large, multi-property instances of OWL data in navigable form to the user.



Search Mindswap.org

Main ontology loaded: http://owl.mindswap.org/2003/ont/owlweb.rdf

Search for object
of type http://owl.mindswap.org/2003/ont/owlweb.rdf#Swapper

More specific: http://owl.mindswap.org/2003/ont/owlweb.rdf#Student

Searching for *any* http://owl.mindswap.org/2003/ont/owlweb.rdf#Student...

- http://owl.mindswap.org/2003/ont/owlweb.rdf#AlumniGraduateStudent
    - *of type* Student
- http://owl.mindswap.org/2003/ont/owlweb.rdf#UndergraduateStudent
    - *of type* Student
- Some Instance
    - *of type* Student
    - Properties
        - dcms:degreeFrom, foaf:workInfoHomepage, foaf:name, foaf:mbox, dcms:phdFrom, rdf:type
    - Connected to
        - dcms:Swapper, dcms:Student, dcms:GraduateStudent, http://www.cs.umd.edu/~hendler, http://www.mindswap.org/~kettler, Brian Kettler, mailto:kettler@email.address, http://www.cs.umd.edu/~hendler, dcms:AlumniGraduateStudent

Figure 2: MIND SWAP Search Page

7. CONCLUSIONS

Lisp fulfills its typical role of the "powerful backend" in many Semantic Web applications like OilEd and the web-services planning system. In the former, it is invoked to perform the reasoning, while in the latter the planning. Lisp's expressivity and power naturally lends it to these computationally intense tasks. As argued earlier, this is due in part to Lisp's decades of experience with conventional AI problems such as reasoning and planning.

However, in both OilEd and the web-services planning system, less computationally intense implementation aspects such as the interface to the semantic data are abstracted away from Lisp and left to Java. Our search page illustrates that this does not have to be the case. Lisp easily fulfills the role of both the interface (via Wilbur and AllegroServe) and the backend (reasoning via OWLLisaKB) in the search page. Moreover, in systems like the web-services planner, using a native-Lisp interface to the semantic data (e.g., Wilbur) in parallel with SHOP2 eliminates the wasted computation time currently spent on mapping SHOP2 symbols to their respective resources on the web, thus improving performance. We are currently experimenting with such system.

Finally, it is easier and more logical to try to bridge the gap between Lisp's web capabilities and newer programming languages than attempt to reimplement decades of AI research in the former. Systems like Wilbur have made advancements on this front by providing a solid foundation for supporting the rapidly evolving Semantic Web standards, such as RDF, RDFS and OWL. Readily available Lisp implementations of reasoning, planning, rule-based expert systems and other AI techniques make it a perfect match for use in Semantic Web applications. Lisp vendors like Franz are also making progress in this direction with their new interfaces for SOAP and WSDL. For progress to continue, such web standards must be actively supported for Lisp to play a key role in Semantic Web application development.

8. ACKNOWLEDGEMENTS

Thanks to David Silber and Jennifer Golbeck for patiently reviewing drafts of this paper and providing valuable feedback.

9. REFERENCES

[1] Portable allegroserve, August 2001. http://portableaserve.sourceforge.net/.

[2] The Common Lisp Wiki, August 2003. http://www.cliki.net/CLiki.

[6]This is implemented using AServe's `publish` function.

[3] DAML-S Coalition: Anupriya Ankolekar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David Martin, Drew McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry Payne, and Katia Sycara. DAML-S: Web service description for the Semantic Web. *Lecture Notes in Computer Science*, 2342:348–??, 2002. http://link.springer-ny.com/link/service/series/0558/papers/2342/23420348.pdf.

[4] Daniel Barlow. Araneida: a free CL web server, August 2003. http://araneida.telent.net/docs/index.html.

[5] Daniel Barlow. The SBCL Project - Steel Bank Common Lisp, August 2003. http://www.sbcl.org.

[6] Sean Bechhofer. The DIG Description Logic Interface: DIG/1.0. February 2003. http://dl-web.man.ac.uk/dig/2003/02/interface.pdf.

[7] Sean Bechhofer, Ian Horrocks, Carole 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*, number 2174 in Lecture Notes in Computer Science, pages 396–408, Vienna, September 2001. Springer-Verlag.

[8] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic Web. *Scientific American*, 284(5):34–43, May 2001. http://www.sciam.com/2001/0501issue/0501berners-lee.html.

[9] Jeremy J. Carroll and Jos De Roo. *OWL Web Ontology Language Test Cases*, May 2003. http://www.w3.org/TR/owl-test/.

[10] World Wide Web Consortium. Extensible markup language (XML) 1.0 (second edition) – W3C recommendation. Available at http://www.w3.org/TR/2000/WD-xml-2e-20000814, 2000.

[11] Franz, Inc. *jLinker - A Dynamic Link between Lisp and Java*, February 2002. franz.com/support/documentation/6.2/doc/jlinker.htm.

[12] Franz, Inc. *A SOAP 1.1 API for Allegro CL*, July 2003. franz.com/support/documentation/6.2/doc/soap-client.htm.

[13] Franz, Inc. AllegroServe - a Web Application Server, May 2003. http://opensource.franz.com/aserve/.

[14] Paul Graham. Lisp in Web-Based Applications, April 2001. http://yahoo.com/lib/paulgraham/bbnexcerpts.txt.

[15] Volker Haarslev and Ralf Moller. Description of the RACER System and its Applications. In *Proceedubgs International Workshop on Description Logics (DL-2001)*, pages 1–3, August 2001. http://kogs-www.informatik.uni-hamburg.de/ moeller/papers/DL-2001-Racer.ps.gz.

[16] James Hendler. Web Ontology Status, May 2003. Available at http://www.w3.org/2003/Talks/0522-webont-hendler/.

[17] Ian Horrocks. The FaCT system. In Harrie de Swart, editor, *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX-98)*, volume 1397 of *LNAI*, pages 307–312, Berlin, may 1998. Springer.

[18] 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*, 2003. To appear.

[19] Aditya Kalyanpur. SMORE: Semantic Markup, Ontology and RDF Editor, November 2002. http://mindswap.org/ãditkal/editor.shtml.

[20] Yarden Katz and Bijan Parsia. OWLLisaKB: A rule-based OWL reasoner, July 2003. http://www.mindswap.org/k̃atz/OWLLisaKB.

[21] Grecia Lapizco-Encinas. Photostuff, August 2003. http://www.mindswap.org/g̃lapizco/PS.shtml.

[22] Ora Lassila. Enabling Semantic Web Programming by Integrating RDF and Common Lisp. 2002.

[23] Ora Lassila. Taking the RDF model theory out for a spin. *Lecture Notes in Computer Science*, 2342:307–??, 2002. http://link.springer-ny.com/link/service/series/0558/papers/2342/23420307.pdf.

[24] Ora Lassila. Wilbur: Nokia's RDF Toolkit for CLOS. 2002. http://wilbur-rdf.sourceforge.net/docs/.

[25] Ora Lassila. Wilbur: Nokia's RDF Toolkit for CLOS, 2003. http://wilbur-rdf.sourceforge.net/docs/.

[26] J. C. Mallery. A common lisp hypermedia server. In *Proc. 1st Int. Conf. on the World-Wide Web.*, pages ?–?, May 1994.

[27] MINDSWAP, University of Maryland. The MINDSWAP Website, 2003. http://owl.mindswap.org.

[28] Dana Nau, Yue Cao, Amnon Lotem, and Hector Munoz-Avia. SHOP: Simple hierarchical ordered planner. Technical Report CS-TR-3981, University of Maryland, College Park, January 1999.

[29] onShore Development, Inc. Lisp Software. http://alpha.onshored.com/lisp-software/#imho.

[30] onShore Development, Inc. WebCheckout, 2003. http://www.onshored.com/.

[31] Evren Sirin. Web Service Composer, 2003. http://www.mindswap.org/ẽvren/composer/.

[32] Evren Sirin, Bijan Parsia, and Ron Alford. Pellet OWL Reasoner, August 2003. http://www.mindswap.org/2003/pellet/.

[33] Michael K. Smith, Raphael Volz, Deborah McGuiness, and Christopher Welty. Web ontology language (OWL) guide version 1.0. Technical report, W3C World Wide Web Concortium, 2002. http://www.w3.org/TR/2002/WD-owl-guide-20021104/.

[34] Frank van Harmelen, Peter F. Patel-Schneider, and Ian Horrocks. Reference description of the DAML+OIL (March 2001) ontology markup language. Technical report, W3C, March 2001. http://www.daml.org/2001/03/reference.html.

[35] Matt Westhoff and Ross Baker. MailSMORE: A modification to the Semantic Markup, Ontology and RDF Editor, 2002. http://www.mindswap.org/r̃oark/mailSMORE.shtml.

[36] World Wide Web Consortium. Resource Description Framework (RDF) model and syntax specification. Technical report, World Wide Web Consortium, February 1999.

[37] Dan Wu, Evren Sirin, James Hendler, Dana Nau, and Bijan Parsia. Automatic Web Services Composition Using SHOP2. In *The 13th International Conference on Automated Planning and Scheduling - ICAPS2003*, June 2003. http://www.isi.edu/info-agents/workshops/icaps2003-p4ws/papers/wu-icaps2003-p4ws.pdf.

[38] David E. Young. *LISA - Intelligent Software Agents for Common Lisp*. http://lisa.sourceforge.net.