AFRL-IF-RS-TR-2004-274
Final Technical Report
October 2004

SHIFTING THE COMPUTATIONAL PARADIGM

Franklin W. Olin College of Engineering

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. L237

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2004-274 has been reviewed and is approved for publication.

APPROVED: /s/
WAYNE A. BOSCO
Project Engineer

FOR THE DIRECTOR: /s/
JAMES A. COLLINS, Acting Chief
Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	OCTOBER 2004	Final Mar 01 – Feb 04

4. TITLE AND SUBTITLE
SHIFTING THE COMPUTATIONAL PARADIGM

5. FUNDING NUMBERS
C - F30602-01-2-0512
PE - 62301E, 63760E
PR - TASK
TA - 00
WU - 17

6. AUTHOR(S)
Lynn Andrea Stein

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Franklin W. Olin College of Engineering
Olin Way AC-312
Needham Massachusetts 02492

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFTB
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2004-274

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Wayne A. Bosco/IFTB/(315) 330-3578/ Wayne.Bosco@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

This contract concerns the development of semantic web infrastructure and tools through the DARPA Agent Markup Language (DAML) program. (1) Language development. Stein (PI) was original coauthor of the language documents that became the current World Wide Web Consortium (W3C) Recommendation for a semantic web language, OWL. (2) Semantic web services and agents. Olin College worked extensively with members of the DAML-S (web services) team to develop an agent-based understanding of semantic web services. The result, described in joint publications, led to a re-envisioning of web services as active participants in a dynamic environment. (3) Document lifecycle tools. Stein and her group developed a set of semantic web based tools for collaborative document authoring and document lifecycle management. These tools were deployed at Olin College and on the DAML website at daml.org. (4) Semantic web maturity assessment. This group evaluated the suitability of existing DAML and semantic web tools for application construction, including a maturity assessment that fed into the DAML program decision to focus on producing a mature tool suite. (5) Meaning on the semantic web. The semantic web marries knowledge representation to massively distributed infrastructure. This project analyzed the failures of traditional centralized, objectivist approaches to meaning in this context.

14. SUBJECT TERMS
Semantic Web, Document Lifecycle Infrastructure, Knowledge Representation, World Wide Web

15. NUMBER OF PAGES
43

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Tables

1 Summary

This contract concerns the development of semantic web infrastructure and tools through the DARPA Agent Markup Language (DAML) program. The contract to Olin College was initiated in March of 2001. However, Dr. Stein (the PI) began involvement with the DAML program through a separate (earlier) contract with MIT and this report includes that work as it is integrally related to the effort under this contract.

Efforts under this contract fall into five distinct categories:

1. Language development. Stein was original co-author of the language documents that eventually became the current World Wide Web Consortium (W3C) Recommendation for a semantic web language, OWL; her work on this aspect continued throughout this contract.

2. Semantic web services and an agent-based view of the web. This group worked extensively with members of the DAML-S (DAML services) team to develop an agent-based understanding of semantic web services. The results – described in joint publications – led to a re-envisioning of web services as active participants in a dynamic environment, informing further development of web services infrastructure.

3. Document lifecycle tools. Olin College developed a set of semantic web based tools for collaborative document authoring and document lifecycle management. These tools were deployed at Olin College and on the DAML website at daml.org.

4. Semantic web maturity assessment. During the contract period but most specifically in the summer of 2003, this group evaluated the suitability of existing DAML and semantic web tools for application construction, including a maturity assessment that fed into the DAML program decision to focus on producing a mature tool suite.

5. Meaning on the semantic web. The semantic web is a marriage between knowledge representation and massively distributed infrastructure. Traditional approaches to meaning (drawn from philosophy and artificial intelligence) need to be adapted to this new context.

The body of this report draws on previously published work by the participants in the Olin College DAML effort. In particular, several jointly published papers – listed in the final section of this report – are excerpted here. These papers represent the work of the co-authors listed on those papers as well as the members of the Olin College DAML team.

2 Language Development

DAML infrastructure involved the adaptation of a substantial body of work in knowledge representation to the dynamic needs of a scalable, distributed infrastructure as represented by both the World Wide Web and DOD's own networks. Olin College's DAML team worked closely with the leadership of the World Wide Web Consortium to bridge the gap between these two fields, co-authoring the original DAML-O specification on which DAML+OIL and subsequently W3C's OWL semantic web language were built.

2.1 History of Olin Participation

The Olin College PI, Lynn Andrea Stein, has been involved in various aspects of the DAML program on an ongoing basis since its outset (and even prior to its official kickoff). Stein was a member of the MIT jump-start team that developed the initial straw-man document for DAML language(s), nicknamed DAML 0.5 (http://www.w3.org/2000/07/DAML-0-5) and a participant in the DAML jump-start workshop of July 2000 (http://www.w3.org/2000/07/19-DAML). This was the role from which Dr. Stein initiated Olin College's involvement with the DAML effort.

At the DAML kickoff meeting in August 2000, Dr. Hendler asked Olin College – in the person of Dr. Stein – together with Mr. Connolly of the web consortium and Dr. McGuiness of Stanford/KSL, to take on editorship of the initial DAML ontology language, to be based on the 0.5 document together with further developments from the August PI meeting.

The result of this process was the initial release of DAML-ONT (http://www.daml.org/2000/10/daml-ont.html), the first DAML ontology language. Olin College was a central participant in many of the pieces of this effort and Stein was primary author of the walkthrough (http://www.daml.org/2000/10/daml-walkthru).

The primary role played by Olin College in these earliest phases of this project was Dr. Stein's intervention as one of the only "bilingual" citizens of both the web and AI/KR/logic communities. In this capacity, Olin College served as a catalyst for the conversations that made the semantic web infrastructure built under DAML possible. Since the inception of this program, many individuals from both communities have learned to speak the language of and even to live in the world of the other participant community.

Subsequent to the initial release of DAML-ONT, the language committee evolved into what is now the Joint US/EU ad hoc Agent Markup Language Committee (http://www.daml.org/committee/) , and Olin College – through Stein – became a participant in that committee.

2.2 DAML-ONT: An Ontology Language for the Web

This section provides an annotated walk through an example DAML Ontology. The example ontology demonstrates each of the features in DAML-ONT, the initial

specification for DAML Ontologies. This walkthrough includes enough information to enable a web programmer or person versed in knowledge representation to begin programming on the semantic web.

2.2.1 The Ontology

```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-
rdf-syntax-ns#"

xmlns:rdfs="http://www.w3.org/2000/01/rdf
-schema#"

xmlns="http://www.daml.org/2000/10/daml-
ont#">

<rdf:Description about="">
  <versionInfo>$Id: daml-ont.daml,v 1.2
2000/11/14 06:16:00 connolly Exp
$</versionInfo>
  <imports
resource="http://www.w3.org/2000/01/rdf-
schema"/>
</rdf:Description>

<!-- Terms for building classes from
other classes. -->

<Class ID="Thing">
  <label>Thing</label>
  <comment>The most general class in
DAML.</comment>
</Class>

<Class ID="Nothing">
  <comment>the class with no things in
it.</comment>
  <complementOf resource="#Thing"/>
</Class>

<Property ID="disjointWith">
  <label>disjointWith</label>
  <comment>for disjointWith(X, Y) read: X
and Y have no members
        in common.
  </comment>
  <domain resource="#Class"/>
  <range resource="#Class"/>
</Property>

<Class ID="Disjoint">
  <label>Disjoint</label>
  <subClassOf resource="#List"/>
  <comment>for type(L, Disjoint) read:
the classes in L are
  pairwise disjoint.

  i.e. if type(L, Disjoint), and C1 in L
and C2 in L, then disjointWith(C1, C2).
  </comment>
</Class>

<Property ID="unionOf">
  <label>unionOf</label>
  <comment>
    for unionOf(X, Y) read: X is the
union of the classes in the list Y;
    i.e. if something is in any of the
classes in Y, it's in X, and vice versa.
    cf OIL OR</comment>
  <domain resource="#Class"/>
  <range resource="#List"/>
</Property>

<Property ID="disjointUnionOf">
  <label>disjointUnionOf</label>
  <domain resource="#Class"/>
  <range resource="#List"/>
  <comment>

    for disjointUnionOf(X, Y) read: X is
the disjoint union of the classes in
    the list Y: (a) for any c1 and c2 in
Y, disjointWith(c1, c2),
    and (b) i.e. if something is in any
of the classes in Y, it's
    in X, and vice versa.

    cf OIL disjoint-covered
  </comment>
</Property>

<Property ID="intersectionOf">
  <comment>
    for intersectionOf(X, Y) read: X is
the intersection of the classes in the
list Y;
    i.e. if something is in all the
classes in Y, then it's in X, and vice
versa.
cf OIL AND</comment>
  <domain resource="#Class"/>
  <range resource="#List"/>
</Property>

<Property ID="complementOf">
  <comment>
    for complementOf(X, Y) read: X is the
complement of Y; if something is in Y,
    then it's not in X, and vice versa.
cf OIL NOT</comment>
  <domain resource="#Class"/>
  <range resource="#Class"/>
</Property>

<!-- List terminology. -->

<Class ID="List">
  <subClassOf
resource="http://www.w3.org/1999/02/22-
rdf-syntax-ns#Seq"/>
</Class>

<Property ID="oneOf">
  <comment>for oneOf(C, L) read
everything in C is one of the
     things in L;
     This lets us define classes by
enumerating the members.
  </comment>
  <domain resource="#Class"/>
  <range resource="#List"/>
</Property>

<Class ID="Empty">
  <asClass resource="#Nothing"/>
</Class>

<Property ID="first">
  <domain resource="#List"/>
</Property>

<Property ID="rest">
  <domain resource="#List"/>
  <range resource="#List"/>
</Property>
```

```
<Property ID="item">
  <comment>for item(L, I) read: I is an
item in L; either first(L, I)
    or item(R, I) where rest(L,
R).</comment>
  <domain resource="#List"/>
</Property>


<!-- facets of properties. -->

<Property ID="cardinality">
  <label>cardinality</label>
  <comment>for cardinality(P, N) read: P
has cardinality N; i.e.
    everything x in the domain of P has N
things y such that P(x, y).
  </comment>
  <domain resource="#Property"/>
</Property>

<Property ID="maxCardinality">
  <label>maxCardinality</label>
  <comment>for maxCardinality(P, N) read:
P has maximum cardinality N; i.e.
    everything x in the domain of P has
at most N things y such that P(x, y).
  </comment>
  <domain resource="#Property"/>
</Property>

<Property ID="minCardinality">
  <comment>for minCardinality(P, N) read:
P has minimum cardinality N; i.e.
    everything x in the domain of P has
at least N things y such that P(x, y).
  </comment>
  <domain resource="#Property"/>
</Property>

<Property ID="inverseOf">
  <comment>for inverseOf(R, S) read: R is
the inverse of S; i.e.
     if R(x, y) then S(y, x) and vice
versa.</comment>
  <domain resource="#Property"/>
  <range resource="#Property"/>
</Property>

<Class ID="TransitiveProperty"/>

<Class ID="UniqueProperty">
  <label>UniqueProperty</label>
  <comment>compare with maxCardinality=1;
e.g. integer successor:
       if P is a UniqueProperty, then
       if P(x, y) and P(x, z) then y=z.
       aka functional.
  </comment>
  <subClassOf resource="#Property"/>
</Class>

<Class ID="UnambiguousProperty">
  <label
xml:lang="en">UnambiguousProperty</label>
  <comment>if P is an
UnambiguousProperty, then
       if P(x, y) and P(z, y) then x=z.
       aka injective.
       e.g. if nameOfMonth(m, "Feb")
       and nameOfMonth(n, "Feb") then m
and n are the same month.
  </comment>
  <subClassOf resource="#Property"/>
</Class>


<!-- Terms for restricting properties of
things in classes. -->

<Class ID="Restriction"/>

<Property ID="restrictedBy">
  <label>restrictedBy</label>
  <comment>for restrictedBy(C, R), read:
C is restricted by R; i.e. the
       restriction R applies to c;

       if onProperty(R, P) and
toValue(R, V)
       then for every i in C, we have
P(i, V).

       if onProperty(R, P) and
toClass(R, C2)
       then for every i in C and for all
j, if P(i, j) then type(j, C2).
  </comment>
  <domain resource="#Class"/>
  <range resource="#Restriction"/>
</Property>

<Property ID="onProperty">
  <comment>for onProperty(R, P), read:
   R is a restriction/qualification on
P.</comment>
  <domain resource="#Restriction"/>
  <domain resource="#Qualification"/>
  <range resource="#Property"/>
</Property>

<Property ID="toValue">
  <comment>for toValue(R, V), read: R is
a restriction to V.</comment>
  <domain resource="#Restriction"/>
  <range resource="#Class"/>
</Property>

<Property ID="toClass">
  <comment>for toClass(R, C), read: R is
a restriction to C.</comment>
  <domain resource="#Restriction"/>
  <range resource="#Class"/>
</Property>

<Class ID="Qualification"/>

<Property ID="qualifiedBy">
  <label>qualifiedBy</label>
  <comment>for qualifiedBy(C, Q), read: C
is qualified by Q; i.e. the
       qualification Q applies to C;

       if onProperty(Q, P) and
hasValue(Q, C2)
       then for every i in C, there is
some V
       so that type(V, C2) and P(i, V).
  </comment>
  <domain resource="#Class"/>
  <range resource="#Qualification"/>
</Property>

<Property ID="hasValue">
  <label>hasValue</label>
  <comment>for hasValue(Q, C), read: Q is
a hasValue
   qualification to C.</comment>
  <domain resource="#Qualification"/>
  <range resource="#Class"/>
</Property>


<!-- A class for ontologies themselves...
-->

<Class ID="Ontology">
  <label>Ontology</label>
  <comment>An Ontology is a document that
describes
```

```
       a vocabulary of terms for
communication between
       (human and) automated agents.
  </comment>
</Class>

<Property ID="versionInfo">
  <label>versionInfo</label>
  <comment>generally, a string giving
information about this
       version; e.g. RCS/CVS keywords
  </comment>
</Property>

<!-- Importing, i.e. assertion by
reference -->

<Property ID="imports">
  <label>imports</label>
  <comment>for imports(X, Y) read: X
imports Y;
       i.e. X asserts the* contents of Y
by reference;
       i.e. if imports(X, Y) and you
believe X and Y says something,
       then you should believe it.

       Note: "the contents" is, in the
general case,
       an il-formed definite
description. Different
       interactions with a resource may
expose contents
       that vary with time, data format,
preferred language,
       requestor credentials, etc. So
for "the contents",
       read "any contents".
  </comment>
</Property>

<!-- Renaming -->

<Property ID="equivalentTo"> <!-- equals?
equiv? renames? -->
  <comment>for equivalentTo(X, Y), read X
is an equivalent term to Y.
  </comment>

</Property>

<Property ID="sameClassAs">
       <!--@@RDFS specs prohibits
cycles, but I don't buy it. -->
  <subPropertyOf
resource="#equivalentTo"/>
  <subPropertyOf resource="#subClassOf"/>
</Property>

<Property ID="samePropertyAs">
       <!--@@RDFS specs prohibits
cycles, but I don't buy it. -->
  <subPropertyOf
resource="#equivalentTo"/>
  <subPropertyOf
resource="#subPropertyOf"/>
</Property>

<!-- Importing terms from RDF/RDFS -->

<!-- first, assert the contents of the
RDF schema by reference -->
<Ontology about="">
  <imports
resource="http://www.w3.org/2000/01/rdf-
schema"/>
</Ontology>


<Property ID="subPropertyOf">
  <samePropertyAs
resource="http://www.w3.org/2000/01/rdf-
schema#subPropertyOf"/>
  <subPropertyOf
resource="http://www.w3.org/2000/01/rdf-
schema#subPropertyOf"/>
  <!-- the subPropertyOf is for the
benefit of agents that know RDFS
       but don't know DAML. -->
</Property>

<Class ID="Class">
  <sameClassAs
resource="http://www.w3.org/2000/01/rdf-
schema#Class"/>
</Class>

<Class ID="Literal">
  <sameClassAs
resource="http://www.w3.org/2000/01/rdf-
schema#Literal"/>
</Class>

<Class ID="Property">
  <sameClassAs
resource="http://www.w3.org/1999/02/22-
rdf-syntax-ns#Property"/>
</Class>

<Property ID="type">
  <samePropertyAs
resource="http://www.w3.org/1999/02/22-
rdf-syntax-ns#type"/>
</Property>

<Property ID="value">
  <samePropertyAs
resource="http://www.w3.org/1999/02/22-
rdf-syntax-ns#value"/>
</Property>

<Property ID="subClassOf">
  <samePropertyAs
resource="http://www.w3.org/2000/01/rdf-
schema#subClassOf"/>
</Property>

<Property ID="domain">
  <samePropertyAs
resource="http://www.w3.org/2000/01/rdf-
schema#domain"/>
</Property>

<Property ID="range">
  <samePropertyAs
resource="http://www.w3.org/2000/01/rdf-
schema#range"/>
</Property>

<Property ID="label">
  <samePropertyAs
resource="http://www.w3.org/2000/01/rdf-
schema#label"/>
</Property>

<Property ID="comment">
  <samePropertyAs
resource="http://www.w3.org/2000/01/rdf-
schema#comment"/>
</Property>

<Property ID="seeAlso">
  <samePropertyAs
resource="http://www.w3.org/2000/01/rdf-
schema#seeAlso"/>
</Property>

<Property ID="isDefinedBy">
  <samePropertyAs
resource="http://www.w3.org/2000/01/rdf-
schema#isDefinedBy"/>
```

```
  <subPropertyOf resource="#seeAlso"/>
</Property>

<Property ID="default">
  <label>default</label>
  <comment>default(X, Y) suggests that Y
be considered a/the default
       value for the X property. This
can be considered
       documentation (ala label,
comment) but we don't specify
       any logical impact.
  </comment>
</Property>

<!-- from RDF, left out:
       Bag, Alt: why bother? note that
we can't import
                 the syntax of these into
the DAML namespace if we expect
                 RDF 1.0 parsers to grok.
       predicate, subject, object,
Statement: DAML audience doesn't need
quoting.
-->

<!-- from RDFS, left out
       Container: the motivation for
this, to somehow denote that other
                 element names can be used
with the <li> syntax, is busted.
       ContainerMembershipProperty:
without the <li> syntax, not much use for
this.
       ConstraintResource,
ConstraintProperty: I don't grok these.
 -->


</rdf:RDF>
```

2.2.2 A minimalist survival guide to Resource Description Framework (RDF)

This section is intended for those unfamiliar with RDF. It contains only enough information to enable you to read the annotated DAML markup example, and is not intended as a complete introduction to RDF, XML, or any other of the technologies on which DAML is based [Lassila and Swick, 1999; XML, 1998]

RDF is built on XML, which makes use of **tags** to structure information. Here are some example tags:

```
<aTag>...</aTag>

<anEmptyTag/>

<anotherTag with="an attribute">...</anotherTag>

<aTag>with <anotherTag/> inside it</aTag>

<tags>and<moreTags>and<moreTags>and...</moreTags></more
Tags></tags>
```

The first thing after the open angle bracket is the **tag name** . The whole tag is often referred to by this name. So

```
<Class ID="Animal">
```

is a Class tag.

A tag ends with a closing angle bracket.

Some tags are **start tags** , and have matching **end tags** . The end tag has the same tag name as the opening tag, but begins with </ rather than simply <. For example,

```
<Class ID="Animal">
```

would be closed by the matching end tag
`</Class>`

From an opening tag to its matching closing tag is an **element** . Other tags can be embedded inside the element, but all such embedded elements must be closed inside the enclosing element. So `<rdf:RDF>` would begin an element that runs until an `</rdf:RDF>` . In this case, anything between the `<rdf:RDF>` and the matching `</rdf:RDF>` would be enclosed in the scope of this element.

An alternate way to specify an element that has nothing between its opening and its closing is to use a single tag with a slash immediately before the closing angle-bracket. Thus, the **self-closing tag**
`<Class ID="Animal"/>`
is the same as
`<Class ID="Animal"></Class>`
with nothing inside it.

An opening tag (or a self-closing tag) may contain things other than the tag name. These things after the tag name are **attributes** . Each attribute has a name, an equal sign, and an attribute value (generally enclosed in double quotes). So, for example, in the Class tag above,
`ID="Animal"`
is the attribute,
`ID`
is the attribute name, and
`"Animal"`
is the attribute value. Attributes are read as properties of the element in which they appear.

An RDF document is a collection of assertions in **subject verb object** (**SVO**) form. Within the obligatory RDF declaration (typically a tag that begins something like `<rdf:RDF` ...), each topmost element is the subject of a sentence. The next level of enclosed elements represent verb/object pairs for this sentence:

```
<Class ID="Male">
  <subClassOf resource="#Animal"/>
</Class>
```

Male is a subclass of Animal.

```
<Class ID="Female">
  <subClassOf resource="#Animal"/>
  <disjointFrom resource="#Male"/>
</Class>
```

Female is a subclass of Animal AND Female is disjoint from Male. The single subject -- Female -- is used to begin each of the verb-object assertions
`<subClassOf resource="#Animal"/>`

and
```
<disjointFrom resource="#Male"/>
```

A few attributes here require explanation.

ID creates a referenceable name, corresponding to the attribute value. So, for example, ID="Male" means that you can refer to Male and mean the thing described by the Class element above. Similarly, Female is a referenceable name.

The **resource attribute** , then, is simply a reference to such a name. In
`<disjointFrom resource="#Male"/>`
the resource attribute is used to indicate that the object of the assertion "Female is disjoint from" is the thing identified with the name Male. Note the prepended # to refer to the name. This indicates a reference to a name within the same containing document, i.e., a **local name** . It is also possible (e.g., by prepending a url before the #) to refer to a name defined elsewhere.

In the above example, each verb tag is self-closing. It is also possible to embed elements inside these verb elements, making objects that are themselves the subjects of subordinate clauses. This causes an alternation of subject verb subject verb ... sometimes called RDF **striped syntax** .

2.2.3 Setting Up Namespaces

DAML-ONT, as of this draft, is written in RDF, i.e., DAML-ONT markup is a specific kind of RDF markup. RDF, in turn, is written in XML, using XML Namespaces and URIs [Lassila and Swick, 1999; XML, 1998; XML Namespaces, 1999; Berners-Lee et al., 1998].

Thus, our example begins with an RDF start tag including three namespace declarations:
```
<rdf:RDF
  xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-
ns#"
  xmlns     ="http://www.daml.org/2000/10/daml-ont#"
  xmlns:daml="http://www.daml.org/2000/10/daml-ont#"
  >
```

So in this document, the `rdf:` prefix should be understood as referring to things drawn from the namespace called `http://www.w3.org/1999/02/22-rdf-syntax-ns#` . This is a conventional rdf declaration appearing verbatim at the beginning of almost every rdf document.[1]

[1] The namespace prefix rdf is arbitrary here; you can call the namespace associated with http://www.w3.org/1999/02/22-rdf-syntax-ns# whatever you want. Since this is the conventional namespace for rdf syntax, rdf is a common way to spell this prefix.
See section 4.2 Same-document References of [Berners-Lee et al., 1998] .

The second declaration says that in this document, unprefixed element names should be understood as referring to things drawn from the namespace called `http://www.daml.org/2000/10/daml-ont#` . This is a conventional DAML-ONT declaration that should appear verbatim at the beginning of the current generation of DAML-ONT documents.

The third declaration binds the daml: prefix to the same namespace name. So, for example, `<Thing/>` and `<daml:Thing/>` refer to the same thing in this document.[2]

If you look at the bottom of this document, you'll see the matching closing tag, `</rdf:RDF>` .

2.2.4 Housekeeping

The first thing we do inside this RDF is to assert that this *is* an ontology.

```
<Ontology about="">
```

This assertion is formulaic; the about attribute will typically be empty, indicating that the subject of this assertion is *this* document.[3]

Then we give a couple properties of this ontology for documentation purposes:

```
<versionInfo>$ Id : daml-ex.daml,v 1.2 2000/10/07
03:21:17 connolly Exp $</versionInfo>
  <comment>An example ontology</comment>
```

followed by...

```
<imports resource="http://www.daml.org/2000/10/daml-
ont"/>
```

Inside the Ontology element, we list any imported ontologies (using imports properties). This particular ontology depends only on the standard DAML ontology (used in the namespace definitions above).

Note that this (imports) tag is an empty element; the same tag starts and ends the element.

```
</Ontology>
```

2.2.5 Defining Classes and Properties

Now, we begin our ontology definitions. First, we define a kind of thing called an animal. To do this, we use a Class tag.[4]

```
<Class ID="Animal">
```

---

[2] Un-prefixed attribute names are not associated with the default namespace name the way un-prefixed element names are.

[3] See myth #4 in Namespace Myths Exploded (Ronald Bourret Mar. 8, 2000 in XML.com).

[4] DAML Class is a synonym for rdfs:Class.

This asserts that there is a Class known as Animal. It doesn't say anything else about what an animal is, etc. It is also not (necessarily) the sole source of information about Animals; we will see below how we can add to a definition made elsewhere.

However, by saying that its ID is Animal, we make it possible for others to refer to the definition of Animal we're giving here. (This is done using the URI of the containing page followed by #Animal .)

```
<label>Animal</label>
 <comment>This class of animals is illustrative of a
number of
 ontological idioms.</comment>
```

These two lines introduce a label -- a brief identifier of the enclosing element, suitable for graphical representations of RDF, etc. -- and a comment -- a natural language (English, in this case) description of the element within which it is included. Neither a label nor a comment contributes to the logical interpretation of the DAML.

```
</Class>
```

There are two types of animals, Male and Female.

```
<Class ID="Male">
 <subClassOf resource="#Animal"/>
</Class>
```

The subClassOf element asserts that its subject -- Male -- is a subclass of its object -- the resource identified by #Animal.[5]

```
<Class ID="Female">
 <subClassOf resource="#Animal"/>
 <disjointFrom resource="#Male"/>
</Class>
```

Some animals are Female, too, but nothing can be both Male and Female (in this ontology) because these two classes are disjoint (using the disjointFrom tag):

Next, we define a property. A property -- or binary relation -- connects two items. In this case, we're defining the parent relation.[6]

```
<Property ID="parent">
```

The property definition begins similarly to the Class: There is a property called parent. Note, however, that this is not a closing tag; there's more to this definition. (There is a matching </Property> tag below.)

```
<cardinality>2</cardinality>
```

[5] DAML SubClassOf is a synonym for rdfs:subClassOf.
[6] DAML Property is a synonym for rdf:Property.

Embedded elements, such as this one, are understood to describe their enclosing elements. So this cardinality element describes the Property whose ID is parent. It says that each thing-that-has-a-parent has two of them.

```
<domain resource="#Animal"/>
```

This element also describes the Property whose ID is parent. It says that the domain [Footnote: rdfs:domain] of the parent relation is Animals. That is, we're defining parent as a thing that an animal has.

We do this by asserting that there's a thing which is a domain -- a description of what kinds of things you can talk about the parents of -- and saying that the domain of this thing (the Property with ID parent) is the resource known as #Animal. What is the resource attribute? It is a reference to something. (Note: ID creates a reference-able description; resource refers to such a description.)

Each of these names -- Person, Man, and Woman -- refers to names in this document, since each reference begins with a #.

```
</Property>
```

That's all we have to say about this Property (whose ID = parent).

Now we'll define a Class with some attributes.

```
<Class ID="Person">
```

This element describes a kind of thing called a Person, and is referenceable as such.

```
<subClassOf resource="#Animal"/>
```

A Person's a kind of Animal (see definition of Animal, above).

The next few lines describe a domain-specific range restriction. To whit, the parent of a Person is also a Person.

```
<restrictedBy>
  <Restriction>
   <onProperty resource="#parent"/>
   <toClass resource="#Person"/>
  </Restriction>
 </restrictedBy>
```

The syntax used here is a cliché, i.e., it is almost always used as shown, except for the name of the resource in the OnProperty element, which will give the name of the Property to be restricted, and the resource associated with the toClass element, which will give the name of the Class to which the Property is restricted.

```
</Class>
```

That's the end of the Person class.

The next several annotations illustrate features of properties:

Father is a property that is a kind of parent property, i.e., x's father is also x's parent.[7] In addition, range is used to ensure that x's father must be a Man, and that x has only one father.

```
<Property ID="father">
 <subProperty resource="#parent"/>
 <range resource="#Man"/>
 <cardinality>1</cardinality>
</Property>
```

Mother is defined similarly but using a variant notation. A UniqueProperty is one with cardinality 1, so we can omit that sub-element from Mother's definition.

```
<UniqueProperty ID="mother">
 <subProperty resource="#parent"/>
 <range resource="#Woman"/>
</UniqueProperty>
```

See also UnambiguousProperty, a property whose object uniquely identifies its subject.

Sometimes, we like to refer to mothers using the synonym mom. The tag equivalentTo allows us to establish this synonymy:

```
<Property ID="mom">
 <equivalentTo resource="#mother"/>
</Property>
```

If x's parent is y, then y is x's child. This is defined using the inverseOf tag.

```
<Property ID="child">
 <inverseOf resource="#parent"/>
</Property>
```

The ancestor and descendent properties are transitive versions of the parent and child properties. We would need to introduce additional elements to enforce these connections.

```
<TransitiveProperty ID="ancestor">
</TransitiveProperty>

<TransitiveProperty ID="descendant"/>
```

The cardinality property is exact. But sometimes we want to bound cardinality without precisely specifying it. A person may have zero or one job, no more:

```
<Property ID="occupation">
 <maxCardinality>1</maxCardinality>
</Property>
```

Classes, too, can be annotated in various ways:

[7] This is rdfs:subProperty.

```
<Class ID="Car">
 <comment>no car is a person</comment>
<subClassOf>
```

The thing that Car is a subClassOf could in principle be specified using a resource= attribute. In this case, however, there is no preexisting succinct name for the thing we want. A car is a kind of non-person. We build this by introducing a new -- anonymous -- Class definition described using the complementOf tag:

```
<Class>
   <complementOf resource="#Person"/>
```

From the inside out: There's a thing that's the complement of Person, i.e., all non-Persons.

```
</Class>
```

That thing is a Class.

```
</subClassOf>
```

That thing -- the Class of all non-Persons -- has a subClass.

```
</Class>
```

The Class with ID Car is the thing that is a subClass of the Class of all non-Persons.

(There's a similar construction from the outside in: Car is a Class that is a specialization of another Class, the Class that's left when you consider everything except Persons).

A Man is a kind of Male Person

```
<Class ID="Man">
 <subClassOf resource="#Person"/>
 <subClassOf resource="#Male"/>
</Class>
```

...and a Woman's a Female Person.

```
<Class ID="Woman">
 <subClassOf resource="#Person"/>
 <subClassOf resource="#Female"/>
</Class>
```

Here is an example of further specifying a previously defined element by using the about attribute. These assertions about the thing described above with ID Person have no more and no less authority than the assertions made within the <Class ID="Person"> element. (Of course, if the two assertions were in different documents, had different authors, etc., they might have different authority.)

In this case, we identify the Class Person with the disjoint union of the Classes Man and Woman. Note that the disjointUnionOf element contains two subelements, the Class Man and the Class Woman. The parseType= "daml:collection" indicates that these subelements are to be treated as a unit, i.e., that they have special RDF-extending meaning within the disjointUnionOf.

```
<Class about="#Person">
 <comment>every person is a man or a woman</comment>
```

```
 <disjointUnionOf parseType="daml:collection">
  <Class about="#Man"/>
  <Class about="#Woman"/>
 </disjointUnionOf>
</Class>
```

We can also define individuals, e.g., Adam:

```
<Person ID="Adam">
 <label>Adam</label>
 <comment>Adam is a person.</comment>
</Person>
```

A Person has a height, which is a Height. (height is a Property, or relation; Height is a Class, or kind of thing.)

```
<Property ID="height">
 <domain resource="#Person"/>
 <range resource="#Height"/>
</Property>
```

Height is Class described by an explicitly enumerated set. We can describe this set using the oneOf element. Like disjointUnionOf, oneOf uses the RDF-extending parsetype="daml:collection".

```
<Class ID="Height">
 <oneOf parseType="daml:collection">
  <Height ID="short"/>
  <Height ID="medium"/>
  <Height ID="tall"/>
 </oneOf>
</Class>
```

Finally, TallMan is the Class defined by intersecting TallThing and Man. The intersectionOf element also uses parsetype="daml:collection".

```
<Class ID="TallMan">
 <intersectionOf parseType="daml:collection">
 <Class about="#TallThing"/>
 <Class about="#Man"/>
 </intersectionOf>
</Class>
```

Finally, we end the rdf:RDF element.

```
</rdf:RDF>
```

2.3 Subsequent Changes to DAML Ontology Languages

The original DAML-ONT document continued to evolve under the guidance of the Joint Committee (including Olin College participation) and was subsequently reissued as DAML+OIL, the DAML program's official ontology language. This in turn was adopted as a starting point for the World Wide Web Consortium's Web Ontology (WebOnt)

Working Group and issued as a W3C Technical Report (http://www.w3.org/TR/daml+oil-reference).

With the establishment of the WebOnt working group at W3C, chaired by Dr. Jim Hendler (original PM of the DAML program), Olin College's language efforts shifted towards the consolidation of the initial DAML+OIL work and its transformation into a viable web tool. Dr. Stein joined the WebOnt working group as an invited expert. The WebOnt working group produced its own ontology language, called OWL, based closely on the original DAML-ONT specification that Olin College PI Stein co-authored, The OWL specification was initially released in July 2002 and finally issued as a W3C Recommendation in February 2004 (http://www.w3.org/TR/owl-ref/). Upon OWL's creation by the WebOnt Working Group (and per order of the DAML PM), OWL was adopted by the DAML program to replace its own official ontology language.

The Web Ontology Language OWL is a semantic markup language for publishing and sharing ontologies on the World Wide Web. OWL is developed as a vocabulary extension of RDF (the Resource Description Framework [RDF, 1999]) and is derived from the DAML+OIL Web Ontology Language, itself derived from DAML-ONT. The next two subsections summarize the major changes made from the original DAML-ONT specification in each of these subsequent language releases.

2.3.1 DAML+OIL Differences

1. Redefinition of Thing (now Thing is the union of Nothing and the complement of Nothing).
2. Addition of localized cardinality restriction.
3. Elimination of restrictedBy.
4. Elimination of qualifications (superceded by restrictions).
5. Clarification of restriction semantics; now restrictions are necessary *and* sufficient.
6. Elimination of cardinality facets on slots.
7. Renaming of certain properties to increase clarity.
8. Elimination of "default".

2.3.2 OWL Differences

1. Removal of qualified number restrictions.
2. Addition of explicitly symmetric properties.
3. Removal of certain ambiguous constructs.
4. Renaming of certain features to increase clarity.

3 Web Services and Agents

Olin College engaged in a significant collaboration with the DAML-S (DAML Services) effort and produced papers bridging our prior work with theirs. As a result of these efforts, we were able to provide feedback that will contribute to the development of an active rather than passive Semantic Web.

This work takes an agent-oriented approach to building the semantic Web, in which we view a system as a collection of agents—human-like actors with beliefs, intentions, and abilities. This approach offers an intuitive way to reason about systems, which makes it easier to build them. Software engineers can reason about large systems using the social skills our species has been honing for thousands of years [Ciancarini and Wooldridge, 2001].

Agent-oriented software engineering emphasizes that the Web is not just about information; it's about services—about getting things done. The Web connects to companies and organizations, people and software programs. Services are a form of behavior, and, according to Alan Turing (the Turing test), behavior is the fundamental attribute of intelligence. By connecting AI agents to these services we are not just giving agents more information; we are giving them more behavior. We are extending their intelligence so that they can affect the real world.

Of course, agent-oriented software engineering is far from trivial. To achieve this vision for the semantic Web we need two things: a methodology for building agents that can safely extend their own behavior, and a way to mark up the semantic Web to support this type of extension.

To engineer extendable agents, we propose using an agent-oriented approach to software engineering called Behavior-Oriented Design (BOD) [Bryson and Stein, 2001a] BOD supplies both modularity—permitting compartmentalization—and mechanisms for intermodule coordination. The markup language we build on is a refinement of DAML-S [Ankolekar et al., 2001]. DAML-S markup on services and BOD modules lets a BOD agent make informed decisions about using them. DAML-S also encodes the BOD reactive plans that combine, control, and arbitrate among these modules. The resulting agents are the new semantic Web's active intelligence.

3.1 Bringing Agents to the Web

DAML-S stems from the Defense Advanced Research Projects Agency's (DARPA's) work to produce the DARPA Agent Markup Language (DAML). DAML is itself an extension of the Extensible Markup Language (XML) and the Resource Description Framework (RDF), which aim to provide better specifications of relationships and ontologies within Web pages. The ultimate goal is to facilitate the automated parsing of Web pages and thus the intelligent use of data on the Web. DAML-S is DAML devoted to describing Web services. It will let users and software agents automatically discover, invoke, compose, and monitor Web resources that offer services, under specified constraints.

Using a formal language like DAML-S offers enormous benefits. Not only does it enable the precise handling of information-retrieval searches and more elaborate types of queries over Web content, but it also opens the door to powerful forms of reasoning about that content [Denker et al., 2001]. Because its designers recognized the problem of combining services, DAML-S already provides nearly all the infrastructure to support our approach. However, understanding the semantic Web as active agent intelligence does require some refinements to the current DAML-S specification.

Any agent design methodology suitable to exploiting discovered services must both emphasize modularity and have some mechanism for coordinating modules. Using BOD, designers develop modular agents in the behavior-based tradition. To coordinate and arbitrate among these modules, designers build explicit, hand-coded reactive plans. These reactive plans encode the agents' (possibly conflicting) goals and their priorities. For example, an agent may have the goal of buying a plane ticket and of retaining a positive balance in a bank account. Either of these goals may in theory have higher priority; BOD's reactive plans let the agent's designer specify which goal does have priority and in what circumstances. Meanwhile, the designer can encode the nitty-gritty of pursuing the goal inside a software module.

The BOD agent architecture is one of a number that combine behavior-based AI and reactive planning [Bryson, 2000]. BOD's agents differ from other hybrid architectures because BOD gives more power and autonomy to the behavior modules and reduces the need for plans to arbitrate among modules that vie for resources. However, our proposals should apply to a range of agent architectures.

3.2 Services, Agents, and Behavior

A Web service is any Web-accessible program or device, and it may or may not affect the real world. For example, a service that searches the Web for research on Haiti changes only the agent's own knowledge state; a service to buy an airline ticket debits the user's charge account and enters that user on the passenger list. Other examples of Web services are a software company providing a patch to fix a program, the police department sending an officer to check a house, or a post office printing an e-mail message and delivering it as surface mail.

A composite service combines individual services in a way that adds value to the user. An example is a travel agency, where a customer specifies the preferred type of trip, and the travel agent selects or assists in selecting specific service providers, such as the airline and hotel. To combine the simpler services of which it is composed, a composite service typically has coordination or arbitration rules that let it prioritize or make tradeoffs, like booking an airline ticket to lock in a good rate before fully investigating hotel options.

An agent is any relatively autonomous actor, typically with

- *goals*, conditions it works to achieve or maintain;
- *intentions*, goals and subgoals that it is currently pursuing,

- *beliefs*, knowledge about the world, which is necessarily limited and possibly inaccurate; and
- *behaviors*, actions it can take.

Most people view agents as consumers of services. However, in employing a service, you are also, in a sense, acquiring new behavior. For example, if you need to paint your house, you can buy a brush or hire a painter. You can use either to paint your house. Although you might not think of a hired painter as part of yourself, the painter helps you achieve your goal. Further, you must alter your own plans to monitor the painter and to accommodate his actions. The painter might even help you store memories: Years later, when you contact him for the paint brand and color, he can supply it. So, as Andy Clark noted in *Being There: Putting Brain, Body and World Together Again* (MIT Press, Cambridge, Mass., 1996), things outside your person may become a part of your agency. This is at least as true for an AI agent as it is for a human.

3.3 Modular Agents and BOD

Modularity simplifies software engineering because it lets designers decompose a complex program into relatively simple modules. A programmer or designer can then develop and debug these modules independently. Modularity underlies object-oriented design as well as the behavior-based approach to AI, which has become at least part of many leading agent-architecture paradigms [Bryson, 2000; Gat, 1998].

Modularity generally simplifies design, but it also creates two formidable challenges. One is decomposition: how many modules and what resources in each? The other is coordination. What if independent components attempt contradictory behavior? For example, what if a module dedicated to charity and a module dedicated to housekeeping both decide to make a major expenditure in the same month? The overall system must have some way to arbitrate between these modules and decide which action will execute.

BOD, like object-oriented design, specifies that modules group actions that require shared variable state. Designers initially specify reactive plans as simple sequences of actions that the agent can be expected to take. BOD provides both a six-step guide for initial decomposition and a cyclic development process for actual implementation. The development process features heuristics for reevaluating which parts of the agent's intelligence the designer should represent in separate behaviors and which in the reactive plans. Consequently, the agent's structure stays as simple as possible while its behavioral complexity increases [Bryson and Stein, 2001a].

Applying BOD to semantic Web development constrains some of this development process. On the Web, services are the behavior modules—black boxes from the perspective of any client agent, either human or artificial. The agent can turn knobs and switches (on a travel service, for example, it can choose a date and destination) but it has no control over how the module actually works. Nevertheless, agents are in many respects just like behavior modules. They provide perceptual information (tell you the price or availability of a flight), perform actions (buy a ticket), and maintain state (remember your itinerary).

3.4 Module Coordination

The most popular way to arbitrate behavior-based modular systems is to incorporate hierarchical reactive plans into system execution [Bryson, 2000; Gat, 1998]. Reactive planning addresses the problem of action selection by looking up the next action based on the current context, in contrast to deliberate or constructive planning, which involve search and means-to-ends reasoning. Reactive plans are established structures that support the look-up process. Hierarchical reactive plans are simple, robust plans, each element of which may itself be another reactive plan.

BOD uses parallel-rooted, ordered slip-stack hierarchical (POSH) reactive plans [Bryson and Stein, 2001b]. At the root of the plan hierarchy are the agent's top-level goals. For example, suppose Mojda has an agent with two jobs: notify her of meetings and search the Web for articles relevant to her research. The agent's plan hierarchy would have two root branches; the higher priority goal (branch) would be monitoring her schedule.

On every program cycle, a POSH agent's coordination module checks to see which root goal it should attend to (as determined by priority, preconditions, and optional scheduling). It then pursues progress toward that goal, as determined by the state of component behaviors and the last action recorded in that branch of the POSH plan.

A coordination module pursues progress toward a goal in several ways. First, a behavior module/Web service may be making progress independently and in parallel to the coordination module. In this case, the coordination module needs to determine merely whether the behavior module is complete. Second, achieving the goal may require the agent to perform actions in a set sequence. In this case, the coordination module recalls its current place within the sequence and triggers the next step. Finally, sometimes a goal requires that the agent perform actions sequentially, but the exact sequencing cannot be determined in advance. In this case, the designer supplies a basic reactive plan—an elaborated sequence that incorporates production-rule-like technology.

3.4.1 How a Basic Reactive Plan Works

A basic reactive plan is an elaboration of a simple sequence that allows for a reactive response to dynamic environments. Execution of a BRP can skip or repeat elements of the sequence as necessary. The final (terminating) element has the highest priority. A precondition guards each element, determining whether that element can execute guards each element. Each program cycle of the behavior-arbitration module executes the highest priority, currently executable element of the currently attended BRP. A BRP step is a tuple of priority, releaser, and action. Each BRP contains the small set of steps associated with achieving a particular goal condition. The releaser for a step is a conjunction of Boolean perceptual primitives that determines whether the step can execute. Each action can be either another BRP or a more primitive plan element.

The releaser and priority determine the order in which plan steps are expressed. If more than one step is operable, the priority determines which step executes. If no step can fire, the BRP terminates. The top priority step of a BRP is often, though not necessarily, a

goal condition. In that case, its releaser recognizes that the BRP has succeeded, and its action terminates the BRP.

3.5 Agent support in DAML-S

The semantic Web is Web intelligence waiting to be discovered and incorporated by intelligent agents. DAML-S has two roles in realizing that vision. First, every service must be described in such a way that an agent can know what it provides and how to interact with it. Second, DAML-S provides support for composite services, combinations of simpler services—or behaviors—and the coordination mechanisms—or reactive plans—used to combine those behaviors.

3.5.1 Composite services

In DAML-S composite services, both behaviors and behavior arbitration are present on the Web. A composite service creates reliable, uniform, higher-level subgoals and specifies services that can partially achieve these subgoals [McIlraith and Son, 2002]. For example, if Mojda decides she needs a vacation, a composite service lets her say, "Buy me the cheapest ticket from here to France." instead of "Purchase AcmeAir Flight 309 Date 20th November."

You can view a composite service as a conventional program, or, more essentially, as another, more powerful service. However, we see an advantage to thinking of a Web service as either an agent in itself, or even as a part of an agent—an extension any agent could apply to itself once it finds and chooses to adopt the service.

One way an agent could interact with a composite service is if the composite service itself behaved as an agent. Suppose userAgent is an agent serving a user. userAgent might discover and enlist a number of compositeServiceAgents to provide a particular service. Before making a final purchase, userAgent can expect the compositeServiceAgents to engage in a negotiation between themselves to select the best offer, perhaps with the userAgent serving as an auctioneer.

An even more compelling scenario is if userAgent can absorb the composite service directly as part of userAgent's intelligence. Because modules and services are relatively autonomous in a BOD agent, this requires only that the designer append the composite service's arbitration to userAgent's reactive plan hierarchy. Thus, in Mojda's case, the userAgent might be given a higher-level goal: "Get Mojda somewhere nice as soon as possible without overdrawing her checking account," but be given very little plan or module support for how to succeed at this task. userAgent might then access the Web and find a composite service that can plan trips. It would then absorb the functionality of the composite service plan into its own ontology—its own goal and plan structure.

The advantage of incorporating the composite service as part of the userAgent is that it gives userAgent a finer granularity of control. For example, userAgent might discover prices available at multiple sites and hold transactions open in each of them before making a decision about which to terminate and which to accept. A userAgent seeking the cheapest possible vacation might exploit two composite services, "Buy me the

cheapest ticket" and "Rent me the cheapest accommodation." The now-augmented userAgent might be able to intervene in the workings of each composite service, altering and pruning each service's search space in light of the information gleaned from the other.

The advantages are even greater if userAgent's designer can encode userAgent in the same formalism as the composite services. In this case, if the userAgent has the capability to test or reason about its own plan structures, it will be able to evaluate composite services in these same terms. Informed and possibly even secure choices among Web service structures would be possible.

3.5.2 Describing services

At the highest level, DAML-S aims to provide more opportunity to automate all aspects of Web-based service provision and use. To meet that goal, it organizes a service description into three conceptual areas [Ankolekar et al., 2002]:

The *profile* describes what the service does. It characterizes the service for advertising, discovery, and matchmaking—the kinds of information service-seeking agents need.

The *process model* tells how the service works, including information about the service's inputs, outputs, preconditions, and effects. It is also important in composing and monitoring processes.

The *grounding* tells how an agent can access a service. Typically, it specifies a communications protocol, such as RPC, or CORBA IDL, and provides details such as port numbers used in contacting the service.

Once it selects a service, the agent uses its process model and grounding to build a message sequence for interacting with the service. The profile and process model are abstract specifications, in that they do not commit to any particular message format, protocol, or Internet address. The grounding provides the concrete specification of these details.

Since the process model specifies the service behavior, it is the foundation of intelligent Web services and is of most interest to BOD. The model includes three types of processes:

- *Atomic* processes are the units of invocation. From the service requester's view, an atomic process executes and returns in a single step.
- *Simple* processes are like atomic processes in that the requester perceives them as having single-step executions. Unlike atomic processes, however, the requester cannot invoke them directly, and they are not associated with a grounding. Simple processes provide abstract views of atomic or composite processes.

- *Composite* processes are constructed from subprocesses, which can be either atomic, simple, or composite. Designers use control constructs to specify the structure of a composite process.

Control constructs are themselves composite, usually being composed of conditions and process components, which in turn can be either processes or control constructs. For example, the control construct If-Then-Else contains a condition and two subprocesses, one of which executes when the condition is true and the other when the condition is false. Table 1 lists the control constructs in the DAML-S's process model.

Table 1. Control constructs in the DAML-S process ontology.

Construct	Description
Sequence	Execute a list of processes in sequential order.
Concurrent	Execute elements of a bag of processes concurrently.
Split	Invoke elements of a bag of processes.
Split + Join	Invoke elements of a bag of processes and synchronize.
Unordered	Execute all processes in a bag in any order.
Choice	Choose between alternatives and execute.
If-Then-Else	If specified condition holds, execute "Then," else execute "Else."
Repeat-Until	Iterate execution of a bag of processes until a condition holds.
Repeat-While	Iterate execution of a bag of processes while a condition holds.

3.6 Implications for DAML-S

Viewing the semantic Web as containing intelligence rather than just knowledge provides a new perspective for semantic Web protocols like DAML-S. We propose a number of DAML-S extensions, and we believe that these insights apply to other semantic Web ontologies as well. We also believe that encoding both agents and services in the same way is the easiest and most thorough way to achieve our vision. Consequently, these recommendations are for supporting userAgents as well as conventional Web services.

3.6.1 Data

Data are key to modularity and agent design, yet they are not currently part of the DAML-S ontology. Some data are integral parts of an agent itself such as the agent's current decision history or its progress in a search. Other data are maintained in service modules, such as airline ticket prices or even the userAgent's itinerary. Data recovery characteristics, particularly after failures or crashes, should be a functional attribute of the DAML-S service profile.

3.6.2 Primitives

Real-time system primitives can behave in two ways. A primitive can compute and return an answer, taking an arbitrarily long time to complete. Or a primitive can trigger a process to run, returning only success or failure in starting the process, leaving the calling program to check whether the process has completed and for any results.

BOD currently uses a hybrid of these approaches. Technically it uses the former (blocked) primitive call, but it expects the module to have an answer ready immediately. Under BOD, behavior modules normally provide an anytime response [Dean and Boddy, 1988]. Basic services in DAML-S should specify their expected return time and values, possibly guaranteeing timeouts if requested.

3.6.3 Sequences

A sequence's behavior depends on the nature of its primitives. BOD's action selection has two sequence types: a trigger sequence, which expects extremely rapid responses from all its elements and executes within a single planning cycle, and an action pattern, which allows for context-checking and reallocation of control priority between every element. Both sequence types abort if one of their elements returns a failure.

DAML-S includes a sequence subtype, but it does not determine whether sequences can be interrupted. Also, in DAML-S, sequence elements themselves can be subprocesses (simple or composite), which implies that the sequence type can only be slow, like a BOD action pattern. DAML-S should incorporate atomic sequencing like BOD's trigger sequences and specify conditions and mechanisms for premature termination.

3.6.4 Basic reactive plans

Often in a dynamic environment, action selection is too nondeterministic for sequences to direct it. Nevertheless, focusing on a particular subset of an action repertoire is a more efficient and effective way to complete tasks. This is what basic reactive plans are for. A BRP provides an organized way to apply a subset of actions until the module achieves a goal. DAML-S does not currently support the expression of a BRP directly, although developers can build BRPs from a repeat-while statement and cascading if-then-elses. For clarity, though, DAML-S should support BRPs directly as a composition construct.

3.6.5 Agent-level control

As we've described, the root of a BOD reactive plan hierarchy is a set of goals that a module checks every time step. A real-time agent requires such a mechanism for monitoring its environment (including itself) to determine whether one of its high-level goals has become urgent and should determine its current intentions. BOD's action selection uses an extended version of the BRP for this purpose [Bryson, 2000; Bryson and Stein, 2001b]. DAML-S should also include an agent-level control construct.

4 Collaborative Document Systems

DOD operations are critically dependent on information management. A substantial part of this grant group's efforts focused on document management ontologies and tools to support information management and decision making, ultimately delivering a toolset to the DAML community on the daml.org web site.

In this effort, Olin College developed and deployed a set of DAML-enhanced document management tools to provide collaborative document development and version tracking functionality. These tools are based on standard COTS systems (Concurrent Versions System (CVS) and wiki) modified for DAML compatibility. In addition to their suitability for information management applications, these tools are appropriate for software development and software process management applications as similar source and life cycle issues arise in that domain.

4.1 A Queryable Document Versioning Repository

Suppose that intelligence analysts are producing intelligence reports about Afghanistan. There are several types of reports with varying review cycles and timeliness requirements. All the reports are archived using a DAML-based system such as the one that we are developing. Using document-life-cycle enhanced DAML tools such as we built, consumers of intelligence could get answers to questions like:

- What is the most recent report on road traffic in Afghanistan?
- How often have we revised our estimates on fuel supplies in the Northeast region?
- How many reports are based on the raw data given in document?
- Which documents have been entered into the Operational Net Assessment database?

The Olin College CVS-to-DAML servlet returns CVS history information for one of the CVS servers. (See http://unicorn.olin.edu:8080/CVStoRDFservlet/CVStoRDF for a demonstration; the output is rather lengthy.) It uses the CVS ontology developed by Mike Dean, online at http://www.daml.org/2001/10/cvslog/cvslog-ont. The code for the servlet is available as a .war file.

Specifically, we instrumented a CVS (document tracking and versioning) system with DAML (subsequently OWL) markup. This allows semantic web access to multiple versions of a single document and to information about document lifecycle, authorship, revision, etc. We then used this instrumented versioning system to back up a wiki – a collaboratively writable web space – with DAML (subsequently OWL) markup. This allows straight web access to shared document manipulation space with semantic web annotation of the document's life history.

Initially, we deployed our semantic web instrumented wiki in a curricular activity at Olin, engaging Olin students and faculty in testing its general as well as specifically pedagogic/educational applications. This launched a more widespread use of wiki technology in the organizational lifecycle and management of Olin College.

Once this early testing was completed, we created a public version of our toolset on the Olin DAML project web site, http://www.daml.olin.edu. Additional testing, especially by W3C and DAML collaborators, led to further revisions and refinements of our toolset.

Finally, our toolset was deployed on the public DAML project site, http://daml.org, in conjunction with the DAML deliverables at that site. Demos were also given informally to members of the DAML community and formally at the fall 2003 DAML PI meeting.

The final tool uses a refined ontology for CVS data (based on work started by Mike Dean of the BBN DAML Integrator team) and a java servlet that uses a Jena query to produce DAML output from the CVS store. The back end performance of this toolset was significantly enhanced as further analysis allowed for optimization of query times of the CVS-to-RDF servlet.[8] Additional collaboration with the DAML integrator and repackaging of the servlet resulted in deployment on the DAML web site and resulting integration with the larger DAML corpus.

4.2 Documents and People: Expanding the Ontology

The CVS DAML servlet makes information about documents and their histories available on the semantic web. In addition, Olin College built and integrated tools to publish and manipulate information about individuals. This allows reasoning about the people involved in creating and maintaining these documents.

Olin's people tools make use of the FOAF (Friend of a Friend) ontology as a basis for people descriptions. To populate this dataset, we developed conversion tools between FOAF and Vcard representations of data about people as well as a tool to scrape Lightweight Directory Access Protocol (LDAP) data from Olin's Microsoft Exchange server and convert it to DAML format.

On the basis of this person-representation infrastructure, we implemented a Semantic Directory Service. This directory is currently deployed in experimental mode at Olin College. The directory is the basis for further applications the first of which is a student-run DVD lending library.

In order to link the CVS data to information about people, we tested two RDF query systems for linking CVS and FOAF data: The Inkling System for University of Bristol, and the Jena system from HP labs. After discovering that Inkling was broken and that Jena would not handle multiple URIs as input, we modified Jena's query front-end to

[8] Naive Jena queries against the full CVS database (270K) had been taking about 20 minutes. However, by adding some Java code to use regular expressions to filter out redundant CVS transactions from our Wiki , the times dropped to almost 6 minutes. Almost all of this time is 'data load time' in Jena. We investigated, but had not identified prior to abrupt funding termination, suitable alternatives to see whether this time can be dramatically reduced. These included either replacing Jena with a different intermediate system or changing the way in which the CVS repository data is converted into DAML.

support multiple URI data sources. The resulting program is able to handle simple compound queries of the CVS and FOAF data.

4.3 Work Abruptly Terminated

Based on the new directions given at the fall 2003 PI meeting, we presented two alternative plans of action. The first involved work on the ease-of-adoption of existing DAML (OWL) tools; the second involved integration of our CVS-to-OWL tool into an open source distribution such as CVSWeb. We did preliminary work in preparing for both of these tasks, but were not able to fully engage with either as funding was abruptly terminated due to program cuts.

5 Semantic Web Infrastructure Assessment

Olin College undertook an effort to involve Olin students in DAML projects and, through this, to understand the ease of adoption of these tools by non-expert users. The results of these experiments were informative both about the potential of the underlying infrastructure – several interesting tools were built – and its tremendous immaturity – each project ran into roadblocks where documentation was inadequate, where tools could not be properly connected, or where infrastructure simply could not deliver the functionality needed.

Over the lifetime of this contract, but especially intensely in the summer of 2003, we evaluated existing DAML (OWL) tools and selected and deployed an infrastructure set on top of which we built several DAML (OWL) applications. These include an institutional directory and a media database that are interlinked through DAML (OWL) to provide heterogeneous access to these datasets. We also brought up a DAML (OWL) community calendar. Much of this development work was done by summer students; their learning curve provided insight into what is needed to make DAML/OWL more accessible to knowledgeable web programmers with minimal KR backgrounds. We presented the results of these experiments at the fall 2003 PI meeting.

Our demonstration at the PI meeting included the institutional directory and media database as well as our CVS-to-DAML toolset. In addition, this demonstration and other input at the PI meeting focused on an assessment of the (im)maturity of the existing DAML tools. This may have been a factor in the PM's decision at that meeting to focus the final year of the program on the refinement and deployability of basic infrastructure tools rather than the continued exploration of cutting edge innovation that would not be as immediately usable by the broader adopting community.

6 Web Semantics

The semantic web requires sound foundations; this is something that the KR community has long understood. But DAML must also be pragmatic and effective; this is something that is integral to the success of the World Wide Web, software development generally, and the DOD above all.

Olin College brought to the DAML project an unusual background in knowledge representation, software development, and web/information management. This enabled the PI to serve as a bridge between the several communities on which DAML development efforts build. Her role as co-PI of the MIT/W3C DAML grant acknowledges this, although primary funding for Olin College's work came through this separate AFRL/IF Rome grant.[9]

As an expert in logic and knowledge representation, Dr. Stein contributed to web semantics development activities through collaborative investigation of new technologies and their theoretical underpinnings. Much of the Olin College group's foundational work was performed through the WebOnt working group, DAML-JC, and our ongoing collaboration with W3C; additional opportunities to engage with European Semantic Web experts spawned initiatives in collaborative semantics.

Foundational projects undertaken by Olin College include: general semantics for a web ontology language, socially grounded semantics, and syntax-aware semantics:

6.1 General Semantics for a Web Ontology Language

In order for semantic web efforts to succeed, they must be based on a well-founded semantics. This aspect of the project combines the long history of formal semantics — and more recent intense efforts in this area within the AI research community — with the complications of a distributed environment like the world wide web. Olin College has contributed to semantic work on DAML-ONT (the original DAML Ontology Language) and its successors DAML+OIL and OWL as well as an agent-based approach to DAML-S (the DAML Services Infrastructure). Olin College worked closely in recent years with the leadership of the World Wide Web Consortium and other members of the DAML project to bridge the gap between these the pragmatic and the philosophically clean.

6.1.1 Sources and Inspirations

The DAML ontology language takes its motivation from many places, most notably the evolving web languages—in particular RDF [Lassila, 98; Lassila and Swick, 1999] (with the embedded XML) and RDFS [Brickley and Guha, 2000]. It is important to be backwards compatible with existing web standard for interoperability with the growing user base of content, tools for these languages, and users who are comfortable with the languages. The semantics developed for DAML-ONT (and subsequently for DAML+OIL and OWL) builds on the work of RDF/S, capturing semantic relations in

[9] The PI continued to receive small funding through a subcontract to the MIT DAML contract.

machine-readable form through more expressive term descriptions along with precise semantics. This is important for many reasons; arguably the most salient is to facilitate intercommunication between agents. While compatibility with web languages was paramount, we also recognized that markup representational needs went beyond what was conveniently expressible in RDF/S. Thus, an extended language and semantics were developed.

The language and semantics of DAML-ONT (and DAML+OIL, OWL) were also influenced by frame-based systems, including knowledge representation languages such as Ontolingua [Farquhar et al., 1997] or KEE. Frame systems have enjoyed acceptance and perceived ease of use by broad populations and have been embraced relatively widespread use [Fikes and Kehler 1985; Karp 1992; Chaudhri et al. 1998]. The goal of our language is to be accessible to the masses and thus it was important to use paradigms that are easy to explain and use.

Finally, DAML-ONT takes motivation from the field of description logics (www.dl.kr.org), which provide a formal foundation for frame-based systems. Some early description-logic based systems include KL-ONE [Brachman-Schmolze, 1985], CLASSIC [Borgida et. al, 1989], and LOOM [MacGregor, 1991], and a more recent example of a description logic-based system is OIL [Fensel et-al, 2000; Bechhofer et al., 2000]. Description logics emphasize clear, unambiguous languages supported by complete denotational semantics and tractable reasoning algorithms. Description logics have been heavily analyzed in order to understand how constructors interact and combine to impact tractable reasoning. See for example, [Donini et al., 1991A and 1991B] for early evaluations. Also, reasoning algorithms have been studied producing knowledge about efficient reasoning algorithms (See [Horrocks and Patel-Schneider, 1999] and [Horrocks et al., 1999] for example). DAML-ONT draws on the general field of research in description logics and, in particular, on the latest description logic: OIL. OIL was designed to be an expressive description logic that is integrated with modern web technology.

The resulting DAML ontology language is a combination of these three building blocks along with influence from KIF—the Knowledge Interchange Format—a first order logic-based proposed ANSI standard, SHOE—Simple HTML Ontology Language [Heflin and Hendler, 2000], and OKBC—Open Knowledge Base Connectivity—a standard applications programming interface for knowledge systems.

6.1.2 A DAML Semantics

In order to fully specify a knowledge representation language, one needs to describe both the syntax and the semantics of the language. The syntax description specifies what strings of characters are legal statements in the language, and the semantic description specifies the intended meaning of each legal statement in the language. The semantics of a representation language can be formally specified in multiple ways. We have chosen here to use the method of specifying a translation of the DAML-ONT language into another representation language that has a formally specified semantics. In particular, we

have specified a translation of DAML-ONT into first-order predicate calculus, which has a well-accepted model theoretic semantics. We specify how to translate a DAML-ONT ontology into a logical theory expressed in first-order predicate calculus that is claimed to be logically equivalent to the intended meaning of that DAML-ONT ontology.

There is an additional benefit to this approach. By translating a DAML-ONT ontology into a logically equivalent first-order predicate calculus theory, we produce a representation of the ontology from which inferences can automatically be made using traditional automatic theorem provers and problem solvers. For example, the DAML-ONT axioms enable a reasoner to infer from the two statements "Class Male and class Female are disjoint" and "John is type Male" that the statement "John is type Female" is false.

The translation of DAML-ONT to first-order predicate calculus is done by a simple rule for translating an RDF statement into a first-order relational sentence, and by including in the translation a pre-specified set of first-order predicate calculus axioms that restrict the allowable interpretations of the properties and classes that are included in DAML-ONT. This creates a set of first-order sentences that include the specific terms in the ontology along with the pre-specified set of axioms restricting the interpretations. The pre-specified set of axioms and the rules for generating the translation of RDF statements into first order sentences are the focus of this presentation since this is the portion that can be leveraged across all DAML-ONT (and RDF/S) ontologies. Since DAML-ONT is simply a vocabulary of properties and classes added to RDF and RDF Schema, and RDF Schema is simply a vocabulary of properties and classes added to RDF, all statements in DAML-ONT are RDF statements and a rule for translating RDF statements is sufficient for translating DAML-ONT statements as well.

We now describe the mapping of DAML-ONT into first-order predicate calculus. A logical theory that is logically equivalent to a set of DAML-ONT descriptions is produced as follows:

- Translate each RDF statement with property P, subject S, and object O into a first-order predicate calculus sentence of the form " (PropertyValue P S O) ".

- Add to this translation the axioms that constrain the allowable interpretations of the properties and classes that are included in RDF, RDF Schema, and DAML-ONT.

Note that it is not necessary to specify a translation for every construct in RDF since any set of RDF descriptions can be translated into an equivalent set of RDF statements (as described in the RDF and RDF Schema specification documents). Thus, the one translation rule above suffices to translate all of RDF and therefore all of DAML-ONT as well.

A notable characteristic of this axiomatization is that it is designed to minimize the constraints on the legal interpretations of the DAML-ONT properties and classes in the resulting logical theory. In particular, the axioms do not require classes to be sets or

unary relations, nor do they require properties to be sets or binary relations. Such constraints could be added to the resulting logical theory if desired, but they are not needed to express the intended meaning of the DAML-ONT descriptions being translated.

6.1.3 Additional Traditional Semantics for DAML languages

The semantics described here takes an axiomatic approach to providing meaning for semantic web languages. An alternate approach, that of model theory, is used by Hayes (2004) to produce a semantics for RDF and by Patel-Schneider et al. for DAML-ONT, DAML+OIL, and ultimately OWL (2004). Olin College participated in the refinement of these efforts through the DAML community, the Joint Committee, and the Web Ontology Working Group processes. These include teleconferences and face-to-face meetings as well as online discussions. The results of these processes are available as W3C technical reports; in particular, the documentation of the formal semantics developed for OWL may be found at http://www.w3.org/TR/owl-semantics/

6.2 Web Pragmatics

In some ways, a traditional truth-theoretic semantics is just a bad fit for certain aspects of human activity. The web – a dynamic interactive community of practice – is increasingly becoming such a place. Really, this is an issue of (traditional, formal) semantics being applicable to well-formulated concepts and specifically to their truth. Formal semantics are often not really about *meaning*, they are really about *truth*. And it is admissible – even conventional – to say that something is ill formed and therefore without (or outside of) semantics. In natural language, something can be *partially intelligible* or *ill-formed but still comprehensible*. As in natural language, the semantic web will ultimately need to honor these shades of grey.

A classical approach to "the meaning of OWL ontologies", would generally intend something like denotational semantics. (This isn't intended to rule out an axiomatic encoding, but simply to call out "what the term refers to"). Web utterances also have (or, one might argue, *only* have) a *different* kind of meaning. The "social meaning..." of a web utterance includes something one might call "effective semantics", i.e., "what work the term can do in the world".

So, for example, an ontology may formally mean one thing but the courts may (in practice, perhaps even incorrectly) use it as the basis for making a distinct legal ruling. The legal ruling may be at odds with the "meaning" in the classical sense of the ontology, but it then becomes part of the "effective semantics" or "social meaning" of the ontology.

The semantic web is stuck with the reality of living in a world with paradox, contradiction, and other logical difficulties, not to mention ambiguity of meaning, socially constructed usage, partial knowledge, temporal embeddedness, retraction, and outright lies. The approach is to design a system that accommodates to the actual activities of real users, but which – as a consequence – may be more concerned with such things as negotiating misunderstandings than expressing things "correctly".

In fact, the formal specification is giving meaning to the notion of "correct usage". But "correct" is sometimes different from "effective" in the philosophical and pragmatic senses. This approach to semantics recognizes the pragmatic reality that, in an infrastructure as diverse, distributed, and decentralized as the web, there will be misuse, abuse, or what Donald Norman and others would just call ***real*** use.

All of the issues that arise in natural language arise are more general to human interaction. We will have them on the web, whether we like it or not. In a classical approach to formal languages, we might call this misuse or abuse. On the web, we have no choice but to recognize it as real use. Any genuine web semantics will need to take this into account. For example, there *will* be contradiction on the web, and it had better not make OWL collapse. There will also be "shades of meaning" and other things that traditional semantics are lousy at capturing. (Tell me about the semantics of beauty....)

These issues are not just restricted to natural languages, either. Programming languages have formal specifications, but implementations still vary and people code with different interpretations of the programming language. (That's why you get dialects or "write once, debug everywhere" or....) To think that we will be able to precisely specify all meanings on the web is to fundamentally misunderstand the web as well as semantics.

Ultimately, web semantics will need to deal with things like 404 and changing web pages and digital signatures. Assertions will be retracted over time; pages will disappear. These are fundamental issues of "doing logic on the web".

6.3 Communally Derived Semantics

Communities develop usages (e.g., of terms) that constrain meanings. Moving from one community to another often means changing the way that a term is used. A traditional model-theoretic or axiomatic semantics cannot account for these communally derived semantics or for the way meaning shifts as one moves across communities, toolsets, or interrelated ontologies. The underlying theoretical infrastructure must be extended to account for these phenomena if we are to have true semantic interoperability.

The semantic web must allow for semantic interoperability without complete ontology agreement. This strategy rests on the idea of *shared grounding*, finding specific points of agreement and building outward from those to allow just enough common semantics. This model flies in the face of traditional model-theoretic semantics but has been of great utility in other domains, including natural language translation (Knight), human-robot interaction (prior work of the PI with Torrance), and web learning (Etzioni).

This approach argues that the effective semantics of a document must be treated separately from its formal semantics as described in the previous section. The social (AKA effective) meaning of OWL is determined by its external context and the ways in which it hooks into that context. The ability to define a formal semantics (as described above) should not be affected.

Socially grounded semantics allow variations between communities of practice. Truths

that are taken for granted by one group may be falsehoods (or merely ambiguous) to another. What is essential is that each group can reason jointly and that joining a group – or merging groups – can be accomplished incrementally through shared grounding.

For example, one community may accept that there are infinitely many integers and that Paris is the capital of France, while to a C programmer int is restricted to 32 bits and, in certain parts of Texas, Paris is the town next door. One response to this, of course, is that what a C programmer means by an int is not the same thing as the integer (and similarly pjh:Paris is different from at least some Texan's notions), but ultimately that is not an entirely satisfactory response.

There is NO definitive objective interpretation of what, e.g., Paris means, and although I'm confident that you can constrain pjh:Paris in ways that would make it clear that texas:Paris is a different thing, the same issues with subtler shading will arise over and over again.

The only way to truly understand web ontologies is in terms of communities of interpretation: social context. The trick is going to be the ability to say something about being in the same social context and therefore using Paris to mean France, not Texas. (Said differently: If the trick involves its being TRUE that Paris can only mean France, the trick is not going to work on the web.)

In general, we are not going to be able to know the *context* of a particular assertion, but we *will* be able to (and need to) say: For the purposes of the following, assume that you are in the same context as the author of this information. Your assumption may be wrong -- people co-opt terminology and misuse it all the time -- but your reasoning will need to be based on this presumed sharing of context. For the most part, that presumed sharing of context will have to be indicated by human beings or webs of trust.

Classical approaches to logic presume there is *a* universe of discourse. Unfortunately, on the web there are many localized universes that partially overlap and slip and slide against one another. That is, in the world of the web one cannot decontextualize *everything* so that there *can be* a single universe behind it all. Anyone who thinks that this is possible is missing the central idea that mutually inconsistent pockets of reasoning happen *all* the time in the world (not to mention within a single human brain, naive theories of belief notwithstanding).

6.4 Syntactically Constrained Semantics

Certain web mechanisms, such as Digital Signature, constrain the meaning of a statement by tying it to a particular syntactic expression. As the semantic web increasingly relies on the meaning of these syntax-constraining expressions, traditional model-theoretic and axiomatic semantics will need to be similarly extended.

One role of syntax in semantics is the need for systems to make signed (as in digital signature) assertions. The signed assertion “A and B” is different from the signed

assertion “B and A”. For example, the digital signature of “peanut butter and jelly” is different from the digital signature of “jelly and peanut butter”. A semantics for the web will need to take this into account.

Syntax – and a syntactically dependent semantics – is also where the asserter lurks. OWL statements aren't (merely) true or false, they're asserted (by an agent or by the resolvent of a url or by a document) or not.

Default reasoning has always been heavily dependent on syntax and, on the web, this will only be more so. Any system that wants to draw a default conclusion should be allowed to do so but must endorse its conclusion. This means that another reasoner can take that fact – e.g., that Frank says so – as reason enough to believe it. Frank doesn't have to explicitly justify it to me, but may, e.g., sign the assertion or a statement of belief so that I can verify that endorsement. And it may be that Frank wants to change his mind, later, or otherwise retract a default, all of which can be done straightforwardly by modifying (time limiting, withdrawing) his signature of endorsement.

This relieves us to some extent of the obligation to have a universally agreed upon means of default reasoning. I can, of course, sign my own default assertions, but if I make defeasible assertions, I cannot expect to be able to universally justify them beyond my signature. (I may have darn good reasons for believing these defaults, but if they're defaults, they're not expressible directly in OWL and I can't produce an OWL justification.) Others may choose to believe what I say simply because ***I*** say it, or they may reject it, or they may even ask me for some (non-standard, non-universal) explanation of my default conclusion. They can’t expect me to produce a formal proof in a universally agreed upon default logic, though.

7 References cited

1. Extensible Markup Language (XML) 1.0. W3C Recommendation. Feb 1998.

2. T. Berners-Lee, R. Fielding, L. Masinter *Uniform Resource Identifiers (URI): Generic Syntax.* IETF Draft Standard. August 1998 (RFC 2396).

3. Namespaces in XML. W3C Recommendation. Jan 1999.

4. O. Lassila, R. Swick, eds. *Resource Description Framework (RDF) Model and Syntax Specification.* World Wide Web Consortium Recommendation, 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222

5. D. Brickley and R.V. Guha, *Resource Description Framework (RDF) Schema Specification 1.0.* W3C Candidate Recommendation. 27 March 2000. http:// www.w3.org/TR/rdf-schema

6. S.A. McIlraith and T.C. Son, "Adapting Golog for the Composition of Semantic Web Services," *Proc. Eighth Int'l Conf. Knowledge Representation and Reasoning*, Morgan Kaufmann, San Mateo, Calif., 2002, pp. 482-493.

7. T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, vol. 284, no. 5, 2001, pp. 34-43.

8. A. Ankolekar et al., "DAML-S: Semantic Markup for Web services," *Proc. 1st Semantic Web Working Symposium*, I.F. Cruz et al., eds., July 2001; http://www.semanticweb.org/SWWS/program/full/SWWSProceedings.pdf.

9. P. Ciancarini and M. J. Wooldridge, eds., *Proc. First Int'l Workshop Agent-Oriented Software Eng.*, vol. 1957, Lecture Notes in Computer Science. Springer-Verlag, Berlin, 2001.

10. J. J. Bryson and L. A. Stein, "Modularity and Design in Reactive intelligence," *Proc.17th Int'l Joint Conf. Artificial Intelligence*, Morgan Kaufmann, San Mateo, Calif., 2001 (a), pp. 1115-1120.

11. G. Denker et al., "Accessing Information and Services on the DAML-Enabled Web," *Proc. 2nd Int'l Workshop Semantic Web*, S. Staab et al., eds., WWW10 Limited, Hong Kong, 2001, pp.67-78.

12. J. J. Bryson, "Cross-Paradigm Analysis of Autonomous Agent Architecture," *J. Experimental and Theoretical Artificial Intelligence*, vol. 12, no. 2, 2000, pp. 165-190.

13. E. Gat, "ESL: A Language for Supporting Robust Plan Execution in Embedded Autonomous Agents," 1998; http://wwwaig.jpl.nasa.gov/public/home/gat/aero97.html

14. J. J. Bryson and L. A. Stein, "Architectures and Idioms: Making Progress in Agent Design," *Proc. 7th Int'l Workshop Agent Theories, Architectures, and Languages*, C. Castelfranchi and Y. Lesperance, eds., Springer-Verlag, Berlin, 2001 (b), pp.15-30.

15. T. Dean and M. Boddy, "An Analysis of Time-Dependent Planning," *Proc. 7th Nat'l Conf. Artificial Intelligence*, AAAI Press/MIT Press, Menlo Park, Calif., 1988, pp. 49-54.

16. R. Fikes and D. L. McGuinness. "An Axiomatic Semantics for DAML-ONT," December 10, 2000. www.ksl.stanford.edu/people/dlm/daml-semantics .

17. R. Fikes and D. L. McGuinness. "An Axiomatic Semantics for RDF, RDF Schema, and DAML+OIL", KSL Technical Report KSL-01-01, Stanford University, 2001; available on-line as http://www.ksl.stanford.edu/people/dlm/daml-semantics/abstract-axiomatic-semantics.html .

18. D. Fensel, I. Horrocks, F. van Harmelen, S. Decker, M. Erdmann, and M. Klein. OIL in a nutshell. In *Proceedings of the European Knowledge Acquisition Conference* (EKAW-2000) , Lecture Notes In Artificial Intelligence. Springer-Verlag, 2000.
19. J. Heflin and J. Hendler, "Semantic Interoperability on the Web," *Proc. Extreme Markup Languages 2000*, Graphic Communications Assoc., Montreal, Canada 2000, pp. 111–120.
20. Horrocks, I., Patel-Schneider, P., `Optimizing description logic subsumption', *Journal of Logic and Computation*, Vol 9(3), pp 267-293, 1999.
21. I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for very expressive description logics. *Logic Journal of the IGPL*, 8(3):239-263, 1999.
22. R. M. MacGregor: Inside the LOOM Description Classifier. *SIGART Bulletin* 2 (3): 88-92 (1991)
23. F. Donini, M. Lenzerini, D. Nardi, and W. Nutt. `The Complexity of Concept Languages', *KR-91*, pp 151-162, 1991.
24. F. Donini, M. Lenzerini, D. Nardi, and W. Nutt. Tractable concept languages', *IJCAI-91*, pp 458-465, 1991.
25. O. Lassila, "Web Metadata: A Matter of Semantics," *IEEE Internet Computing*, vol. 2, no. 4, July/Aug. 1998, pp. 30–37.
26. R. Fikes and T. Kehler, "The Role of Frame-Based Representation in Reasoning", *CACM* 28(9): 904-920 (1985).
27. P. D. Karp, "The design space of frame knowledge representation systems", Technical Report 520, SRI International AI Center; available on line as ftp://www.ai.sri.com/pub/papers/karp-freview.ps.Z
28. V. Chaudhri, A. Farquhar, R. Fikes, P. Karp, and J. Rice; " OKBC: A Programmatic Foundation for Knowledge Base Interoperability" , AAAI 1998.
29. A. Farquhar, R. Fikes, and J. Rice; "The Ontolingua Server: a Tool for Collaborative Ontology Construction ", Intl. Journal of Human-Computer Studies 46, 1997.
30. S. Bechhofer et al., "An Informal Description of OIL-Core and Standard OIL: A Layered Proposal for DAML-O," www.ontoknowledge.org/oil/downl/dialects.pdf
31. A. Borgida, R. J. Brachman, D. L. McGuinness, and L. Alperin Resnick. ``CLASSIC: A Structural Data Model for Objects", Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data , Portland, Oregon, June, 1989, pp. 59--67.
32. R. J. Brachman, J. G. Schmolze, `An Overview of the KL-ONE Knowledge Representation System', Cognitive Science, Vol 9(2), pp 171-216, 1985.
33. P. Hayes. *RDF Semantics*. W3C Recommendation. February 2004. http://www.w3.org/TR/rdf-mt/
34. P. F. Patel-Schneider, P. Hayes, and I. Horrocks. *OWL Web Ontology Language Semantics and Abstract Syntax*. W3C Recommendation. February 2004. http://www.w3.org/TR/owl-semantics/

8 Publications Resulting From This Work

1. Bryson, Joanna, "Embodiment vs. Memetics: Does Language Need a Physical Plant?"" in developmental Embodied Cognition (DECO-2001), R. Pfeifer and G. Westermann, eds. Edinburgh, July 2001. http://www.cogsci.ed.ac.uk/~deco/
2. Bryson, Joanna, *Intelligence by Design: Principles of Modularity and Coordination*, Technical Report (#AITR-2002-003), MIT Artificial Intelligence Laboratory, September 2001.
3. Bryson, Joanna, "Stable Configurations for Cooperation in Complex Agents with Minimal Communication", NASA GSFC/JPL Workshop on Radical Agent Concepts, September 2001 (postponed to January 2002)..
4. Bryson, Joanna J., David Martin, Sheila McIlraith, and Lynn Andrea Stein. "Toward Behavior Oriented Agents in the Semantic Web." *IEEE Computer* 35(11): 48-54. 2002.
5. Bryson, Joanna J., David Martin, Sheila McIlraith, and Lynn Andrea Stein, "Agent-Based Composite Services in DAML-S: The Behavior-Oriented Design of an Intelligent Semantic Web", *Web Intelligence*, Ning Zhong, Jiming Liu, and Yiyu Yao, eds., Springer-Verlag, 2002
6. Bryson, J., and Lynn Andrea Stein, "Modularity and Design in Reactive Intelligence," *Proceedings of the Seventeen International Joint Conference on Artificial Intelligence*, Seattle, Washington, August 2001.
7. Bryson, Joanna and Lynn Andrea Stein, "Building Agents in DAML-S", unpublished white paper. ftp://ftp.ai.mit.edu/pub/users/joanna/bryson-daml.pdf
8. Connolly, Dan, Ian Horrocks, Frank van Harmelen, Deborah L. McGuinness, Peter F. Patel-Schneider, Lynn Andrea Stein, eds., *DAML+OIL (March 2001) Reference Description* W3C Note, 18 December 2001. http://www.w3.org/TR/daml+oil-reference.
9. Dean, M., G. Schreiber, S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, L.A. Stein, *OWL Web Ontology Language 1.0 Reference* W3C Recommendation 10 February 2004 (Previously Dean, M., D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, L.A. Stein, *OWL Web Ontology Language 1.0 Reference* W3C Working Draft 29 July 2002) http://www.w3.org/TR/owl-ref/.
10. McGuinness, Deborah L., Richard Fikes, Lynn Andrea Stein, and James A. Hendler, "DAML-ONT: An Ontology Language for the Semantic Web," *Spinning the Semantic Web*, D. Fensel, J. Hendler, H. Lieberman, W. Wahlster, eds. MIT Press, Cambridge, 2002
11. Stein, Lynn Andrea, Dan Connolly, and Deborah L. McGuinness, eds., *DAML-ONT Initial Release*, http://www.daml.org/2000/10/daml-ont.html .

12. Stein, Lynn Andrea, Dan Connolly, and Deborah L. McGuinness, eds., “Annotated DAML Ontology Markup,” in Lynn Andrea Stein, Dan Connolly, and Deborah L. McGuinness, eds., *DAML-ONT Initial Release*. http://www.daml.org/2000/10/daml-walkthru .

Also:

Invited Plenary Talk by Stein (PI) at KIMAS 03: "Computation as a Community Member", 10/03

Demo of Olin DAML systems at DARPA DAML PI meeting, 10/03

Our CVS toolset is now also hosted at daml.org