

## **Schemat, tools for ontology-driven identity schema mapping**

Mark Wahl, Informed Control Inc.  
mark.wahl@informed-control.com

Last updated 2006 September 9 (E-2006-12-01)

Keywords: directory, identity, schema, ontology, OWL, RDF, LDAP

### **Abstract**

Schemat is a collection of Java functions for translating ontology instances and classes between formats based on the Web Ontology Language (OWL) [1], XML and the Lightweight Directory Access Protocol Data Interchange Format (LDIF) [2]. It is intended to aid in researching the developments of ontologies to represent identity information and the mappings between ontologies, with a goal of enabling the development of standards enabling extensibility of emerging Internet protocols which exchange identity information to support decentralized localization of identity management systems.

### **Introduction**

A problem faced in many deployments of Identity Management software is the multitude of schema definitions used to represent real-world objects in the Identity Management system. This proliferation can result from fixed schema requirements of particular storage systems (directory servers and databases) and applications expecting a particular schema to be present. This problem will become increasingly difficult as Identity Management deployments based on emerging protocols span multiple, loosely-coupled organizations across the Internet.

Traditionally, two techniques used to address this issue in LDAP [3] directories have been the metadirectory and the directory proxy or virtual directory. A metadirectory typically is a client of one or more directory servers or databases, and is responsible for bringing the contents of each of these repositories into synchronization with each other. Metadirectories use configuration files and custom code to describe the translation from one repository's schema to another. (Some implementations convert to intermediate directory or database server's schema). Examples of metadirectories include Zoomit VIA (now the Microsoft Identity Integration Server) [4], the iPlanet Metadirectory [5], and the Novell Identity Manager 2 (formerly DirXML) [6]. The DirXML product was one of the first to use XML technologies for mapping between directory schemas, specifically leveraging XML Style Sheets (XSLT) [7] in the DirXML engine [8]. A virtual directory server is conceptually similar; the main difference is that the transformations occur "upon demand", when a client sends a request to the virtual directory server, rather than in bulk.

A fundamental assumption of metadirectory or virtual directory designs is that the majority of clients relying on the directory or databases have a fixed schema requirement, and that schema elements not understood by those applications can be omitted by the view provided by those applications without significantly affecting the application. This assumption is usually met as many identity management deployments which have extended the schema in their directory servers primarily add optional, "non-key"

attributes which are used only by custom-developed applications or data management tools, and these extensions can be ignored during queries by third-party directory client applications.

A similar technology, though not so widely used for directory mapping, is enterprise application integration (EAI). Products such as the iPlanet Integration Server [9] use XSLT and other XML mapping technologies to isolate a particular application's access to multiple disparate data sources. However, these EAI products are typically designed to be used by custom-developed applications within J2EE or similar frameworks: it is not typical for a commercial directory-enabled product offering to rely upon a EAI server being present to mediate its access to directory information.

With the continuing rate of introduction of new XML-based protocols for the exchange of identity information, a research project into decentralized localization [10] was initiated in early 2005. The goal of this project is to identify data modeling techniques which could be used in developing a generic approach for schema extensibility in these emerging protocols. These techniques are needed to support private communities evolving their own schema which can be supported within widely-available commercial products, without requiring the product vendor to customize the product for each community. This goal cannot be reached in existing or in these emerging protocols, as

- While some protocols treated the identity information as an opaque collection of elements whose interpretation was left up to the recipient of the data, other protocols, such as FOAF [11], had specified data elements as a core part of their information model (e.g., a person's surname and given name), and
- Even when the protocol does not specify specific data elements, the receiving application would need a means of obtaining the meta-data associated with elements of forms it had not already encountered, such as the matching rules to use for comparing values of these elements, their relative display position (e.g., to associate a individual's surname and given name together), and privacy constraints [12].

As these new protocols are split between those which are based on their own XML schemas and those based on RDF, tools were needed to map identity information, such as might be obtained from a directory server's contents, from existing "legacy" formats such as LDIF into XML and into RDF. The *schemat* tools discussed in this technical report contains several functions which are designed to enable experimentation with such mappings, with an end goal of the experiment being the development of standards and mechanisms to enable extensibility of identity schema in emerging protocols.

## Approach

The *schemat* tools are intended to enable researchers to experiment with mapping identity information from one format to another, using constructs from OWL and RDF [13] to describe the mapping. To simplify implementation and debugging, the tools operate on text files containing the directory contents, similar to first generation meta-directories. While this implementation approach is not suitable for on-the-fly transformation of identity information, it is intended that the algorithms developed for use within the

schemat tools can in the future be extracted and used in a self-contained framework for schema extensibility management.

The first function provided by the schemat tools is the creation of a set of OWL individuals which represent the entries in a directory contents. The approach taken for generating individuals from directory contents is as follows:

- An automated transformation is defined from LDAP subschema elements for attribute type and object class definitions into an OWL-Full ontology. The result of this transformation has classes which resemble LDAP subschema elements.
- An automated transformation is defined from LDAP directory entries to a set of OWL individuals, based on the ontology for the schema elements used in those entries.

The modeling of the individuals is an intermediate form chosen to be as close as possible to the source format, to avoid mapping artifacts being introduced early in the translation process. As a result, these individuals in their intermediate form are not directly useful to applications managing OWL individuals, in particular since most applications assume that a particular base ontology, such as FOAF or Higgins [14], is being used.

To convert OWL individuals from one ontology to another, a set of mapping rules are loaded. These rules are defined as individuals in a mapping ontology, which represent the transformation from an element in one ontology to another. In the current implementation, mappings are defined for predicates (properties) only. This mapping is justified as applications managing OWL individuals are anticipated to have a limited set of OWL classes they can handle (e.g., the FOAF specification has only a few OWL classes, with widely-deployed semantics).

At present, these mapping individuals are hand-written statements, which specify the Uniform Resource Identifier (URI) [15] of a source element, the URI of a destination element, and a process for building the destination element. The first implemented mapping is a direct mapping, which is applicable when there is a string literal property as both the source and destination element of a mapping, and the literals have the same syntax and structure. Additional types of mapping are being defined, based on existing experience in meta-directories, to describe relationships between OWL elements, incorporating translations of values, and of combining and splitting properties, by re-encoding data type valued properties as object properties pointing to a temporary object.

Mappings are a specific example of a more general case of ontology commentary: based on the concepts of RDF, it is possible for one ontology to express additional information on the elements of another ontology. Furthermore, it is expected that the commentary on an ontology may come from a different source from the commented-upon ontology, and that the organization managing the commented-upon ontology may not even be aware of the commentaries which exist for it. This is also the case for a mapping, in which a commentary may be developed by a party independent of the party which developed the source ontology or the destination ontology.

One possible way to manage the mappings between ontologies would be to have a single, authoritative database which contains the ontology commentary. While this approach would be valid for a single organization describing its own ontologies, it begins to break down when multiple organizations are involved, in particular for situations where there is federation, or scenarios in which there are multiple identity providers (such as with OpenID [16] or Project Liberty[17]).

Instead, the approach taken in the schemat tools is to have an RDF scutter [18] build a cache database of commentaries derived from OWL and RDF specification documents obtained from the Internet. This database could be populated with well-known schemas, such as those of the IETF LDAP RFCs, FOAF, Higgins and other protocols.

The database would represent a graph indicating the relationships between the elements of each ontology. Establishing a mapping would consist of building a path (at lowest cost) from the source element to a destination element. This approach has already been pioneered in some meta-directory deployments. However, some new difficulties arise in the current model:

- The "best" destination properties are not always known in advance: there may be multiple potential destination schemas within the capabilities of an application. For example, an LDAP directory service might support the `inetOrgPerson` and `organizationalPerson` object classes equally, and there is no preference to creating the `inetOrgPerson` subclass. In a Higgins environment, there may be multiple OWL classes which are subclasses of `higgins:DigitalSubject`, all of which are suitable for the representations of a person as a digital subject. Furthermore, some protocols may support a choice between the tunneling of data in legacy through them or the use of the native information model of that protocol, resulting in mappings which are the lowest 'cost' of transformation, but not the most widely usable to other applications.
- Ontologies may be present in the database which provide invalid, recursive or useless potential mappings.
- A scutter-based approach may need to be combined with an authentication system for production deployments, in order to prevent the introduction of malicious commentary into the database.

In the current implementation, only single-hop paths are used, although a generic cost-based graph traversal algorithm could later be added.

One area of note in mapping of OWL individuals is the transformation of object properties. For example, mapping a FOAF individual of the `foaf:Person` OWL class into a Higgins-compatible OWL class (one that is a subclass of `higgins:DigitalSubject`) would entail mapping the values of a `foaf:knows` property of that person into the same or another appropriate Higgins-compatible class. In some cases, it might be necessary to retrieve for transformation purposes arbitrary OWL entities from across the Internet in order to complete the transformation, which would have significant performance implications. In the current implementation, it is assumed that all of the related objects are present in a schemat interactions set (described in the Implementation section below).

## RDF Models

The first issue addressed in the development of *schemat* was the representation of existing directory entries and schema in an OWL-compatible RDF model. There are several representation approaches for a directory entry, including:

- (1) the entry is represented as an OWL individual, and the class of the individual is specified as an OWL class representing that it is an entry in the source information model (e.g., LDAP)
- (2) the entry is represented as an OWL individual, and the class of the individual is specified as an OWL class derived from the object class (or classes) of the entry
- (3) each entry is represented as an OWL class

Representation approach (1) was chosen for this project, as this provides the maximum possible fidelity to the source input format, and is intended to be suitable to support formats which do not have a concept of object class. For comparison, approach (2) was adopted by Mr. Dietzold of the University of Leipzig Institute for Medical Information, Statistics and Epidemiology for his implementation of mapping LDAP entries to OWL, and is described in the "Related Work" section below.

The ontology <http://www.ldap.com/1/schema/ldapv3.owl> contains the definitions used to represent LDAP schemas and entries in RDF/OWL. It defines three OWL Classes: *AttributeType*, *ObjectClass* and *Entry*, and the properties listed in the table below.

Property	Domain	Range
<i>Entry_Dn</i>	<i>Entry</i>	XMLSchema#normalizedString
<i>Entry_Has_Attribute_Name</i>	<i>Entry</i>	XMLSchema#token
<i>AttributeType_Name</i>	<i>AttributeType</i>	XMLSchema#token
<i>ObjectClass_Name</i>	<i>ObjectClass</i>	XMLSchema#token
<i>AttributeType_NumericOid</i>	<i>AttributeType</i>	XMLSchema#token
<i>ObjectClass_NumericOid</i>	<i>ObjectClass</i>	XMLSchema#token
<i>AttributeType_Superior</i>	<i>AttributeType</i>	<i>AttributeType</i>
<i>ObjectClass_Superior</i>	<i>ObjectClass</i>	<i>ObjectClass</i>
<i>ObjectClass_Must</i>	<i>ObjectClass</i>	<i>AttributeType</i>
<i>ObjectClass_May</i>	<i>ObjectClass</i>	<i>AttributeType</i>

Ontologies are automatically generated for the Internet Request-For-Comment documents which include widely-used LDAP schemas: RFC 2079 [19], RFC 2798 [20], RFC 4512 [21], RFC 4519 [22], and RFC 4524 [23]. The URIs for these ontologies are of the form <http://www.ldap.com/1/schema/rfcNNNN.owl>, and contain class and property definitions for each attribute type and object class definition in the schema.

For each attribute type definition encountered in a directory schema, an OWL class is generated, that is a subclass of `ldapv3:AttributeType`. The URI for this class is derived from the URI of the source ontology, with a fragment of `AttributeType_numericoid` added.

For each attribute type definition, an OWL property is also generated, with a URI containing a fragment of `numericoid`. In the current implementation, these properties are `owl:DatatypeProperty`, with a string literal value, although in the future other literal types as well as `owl:ObjectProperty` might be used for some attributes.

For each object class definition, an OWL class is generated, that is a subclass of `ldapv3:ObjectClass`. The URI for this class is derived from the URI of the source ontology, with a fragment of `ObjectClass_numericoid` added.

For example, the following LDAP definitions in an "RFC NNNN":

```
attributeTypes: ( 1.2.3.4 NAME 'example' )
objectClasses: ( 1.2.3.5 NAME 'exObject' SUP 2.5.6.0 MUST ( 1.2.3.4 ) )
```

would generate OWL-Full definitions similar to the following:

```
<owl:Class rdf:about="&rfcNNNN;#AttributeType_1.2.3.4">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
  <rdfs:subClassOf rdf:resource="&ldapv3;#AttributeType"/>
  <rdfs:subClassOf><owl:Restriction>
    <owl:onProperty rdf:resource="&ldapv3;#AttributeType_Numericoid"/>
    <owl:hasValue rdf:datatype="http://www.w3.org/2000/01/rdf-schema#Literal">1.2.3.4</owl:hasValue>
  </owl:Restriction></rdfs:subClassOf>

  <rdfs:subClassOf><owl:Restriction>
    <owl:onProperty rdf:resource="&ldapv3;#AttributeType_Name"/>
    <owl:hasValue rdf:datatype="http://www.w3.org/2000/01/rdf-schema#Literal">example</owl:hasValue>
  </owl:Restriction></rdfs:subClassOf>

  <rdfs:label rdf:datatype="http://www.w3.org/2000/01/rdf-schema#Literal">example</rdfs:label>
</owl:Class>

<owl:DatatypeProperty rdf:about="&rfcNNNN;#1.2.3.4">
  <rdfs:domain rdf:about="&ldapv3;#Entry"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<owl:Class rdf:about="&rfcNNNN;#ObjectClass_1.2.3.5">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
  <rdfs:subClassOf rdf:resource="&ldapv3;#ObjectClass"/>
  <rdfs:subClassOf><owl:Restriction>
    <owl:onProperty rdf:resource="&ldapv3;#ObjectClass_Numericoid"/>
    <owl:hasValue rdf:datatype="http://www.w3.org/2000/01/rdf-schema#Literal">1.2.3.5</owl:hasValue>
  </owl:Restriction></rdfs:subClassOf>
```

```
<rdfs:subClassOf><owl:Restriction>  
<owl:onProperty rdf:resource="&ldapv3;#ObjectClass_Name"/>  
<owl:hasValue rdf:datatype="http://www.w3.org/2000/01/rdf-schema#Literal">exObject</owl:hasValue>  
</owl:Restriction></rdfs:subClassOf>
```

```
<rdfs:label rdf:datatype="http://www.w3.org/2000/01/rdf-schema#Literal">exObject</rdfs:label>
```

```
<rdfs:subClassOf><owl:Restriction>  
<owl:onProperty rdf:resource="&ldapv3;#ObjectClass_Superior"/>  
<owl:hasValue rdf:resource="&rfc4512;#ObjectClass_2.5.6.0"></owl:hasValue>  
</owl:Restriction></rdfs:subClassOf>
```

```
<rdfs:subClassOf><owl:Restriction>  
<owl:onProperty rdf:resource="&ldapv3;#ObjectClass_Must"/>  
<owl:hasValue rdf:resource="&rfcXXX;#AttributeType_1.2.3.4"></owl:hasValue>  
</owl:Restriction></rdfs:subClassOf>  
</owl:Class>
```

It is necessary for the resulting ontologies to be in OWL-Full rather than the OWL-DL or OWL-Lite subsets, since this encoding causes classes to have properties that make references to other classes, and OWL does not have the concept of defining an object property to a class other than by making the class also be treated as an individual.

To support FOAF, the FOAF RDF schema (with namespace <http://xmlns.com/foaf/0.1/>) is used, which defines a class Person to represent person entries. Mappings to other FOAF classes are not implemented at present, and could be added later.

To support Higgins, the Higgins RDF schema (with namespace <http://www.eclipse.org/higgins/ontologies/2006/5/higgins>) is used, which defines a class DigitalSubject. This class, however, does not have any properties other than a uniqueidentifier. The specification of OWL classes which are subclasses of DigitalSubject and support the properties derived from mapping from other well-known schemas (such as inetOrgPerson or the FOAF Person) is an ongoing process, and it is anticipated that many such ontologies will be defined.

To represent mappings, the ontology <http://www.ldap.com/1/schema/mapping.rdf> is used. This ontology in the current implementation defines two properties: predicate-mapping and predicate-direct-mapping. It is anticipated that other properties will be added as needed to support other forms of mapping which become useful.

The predicate-mapping is an "abstract base" of mappings from one OWL property to another. A sub-property of this, predicate-direct-mapping, indicates that two properties have the same value (either an object or datatype). For example, a direct mapping from a property for the RFC 4519 attribute "sn" with OID 2.5.4.4 to the FOAF property family\_name can be expressed as follows:

```
<rdf:Description rdf:about="http://www.ldap.com/1/schema/rfc4519.owl#2.5.4.4">  
  <mapping:predicate-direct-mapping rdf:resource="http://xmlns.com/foaf/0.1/family_name"/>  
</rdf:Description>
```

## Implementation

Schemat is implemented in the Java language, and is organized in the following set of packages:

- |   |   |
|---|---|
| • com.informed_control.schemat.i.common         | interfaces for Schemat mapping classes              |
| • com.informed_control.schemat.i.scanner        | interfaces for scanning a schema file               |
| • com.informed_control.schemat.i.index          | interfaces for managing a keyword index             |
| • com.informed_control.schemat.a.common         | abstract base classes for the Schemat I/O methods   |
| • com.informed_control.schemat.a.impl           | implementation of Schemat I/O                       |
| • com.informed_control.schemat.a.impl.owlbindep | common implementation of an OWL reader              |
| • com.informed_control.schemat.a.scanner        | scans a schema file and extracts keywords           |
| • com.informed_control.schemat.a.index          | manages a keyword index using Lucene                |
| • com.informed_control.schemat.a.utils          | utility functions not tied to schemas or ontologies |
| • com.informed_control.schemat.a.web            | servlets providing a web interface to the index     |

The schemat implementation relies upon these third-party components:

- HP ARP2 RDF parser SDK [24]
- OWL API SDK [25]
- Mozilla LDAP SDK [26]
- Apache Lucene search engine [27]
- Apache Jakarta HttpComponents SDK [28]
- Servlet API [29]

The first class typically used in a schemat application is `SchematManager`, which is responsible for creating an `ISchematInteractions`, which encapsulates the set of RDF triples being generated by an `ISchematInputSource` and processed by an `ISchematListener`.

```
class SchematManager {
    static SchematManager getInstance();
    ISchematInteractions startInteractions();
}

interface ISchematInteractions {
    void load(ISchematInputSource s);
    void addEventListener(ISchematListener l);
    void complete();
}
```

There are three input sources: to read from LDIF (using the Netscape Mozilla LDAP SDK LDIF parser), to read from RDF (using HP ARP2), and to read from OWL RDF (using the OWL API).

```
class LdifInputSource implements ISchematInputSource {
    static LdifInputSource newInstanceForLDIF (InputStream is, // from *.ldif
                                              URI base);

    void addDirSchemas(Vector v);
    Vector getDirSchemas();
}
```

```
class RdfInputSource implements ISchematInputSource {
    static RdfInputSource newInstanceForRDF(InputStream is, URI base);
    static RdfInputSource newInstanceForRDF(Reader r, URI base);
}

class OwlInputSource implements ISchematInputSource {
    static OwlInputSource newInstanceForOWL(Reader r, URI base);
}
```

The `LdifInputSource.getDirSchemas()` method returns a `Vector` of `IDirSchema`, consisting of those schemas provided by an earlier call to `addDirSchemas()` and an `IDirSchema` generated by parsing any `attributeTypes` and `objectClasses` values in the LDIF input source.

```
interface IDirSchema {
    IDirSchemaElement findSchemaByOid(String oid);

    IExIterator iterateAttributeTypes();

    IExIterator iterateObjectClasses();

    ISchematInputSource getInputSource();
}
```

The `OwlOntologyOutput` listener writes to the output stream an OWL/RDF file containing the classes in the `ISchematInteractions`. Additional listener classes are defined which implement output of individuals into a particular protocol's format, currently FOAF (based on RDF and using the class `foaf:Person`), Higgins (also based on RDF but using classes subclassed from `higgins:DigitalSubject`) and LDIF.

```
class OwlOntologyOutput implements ISchematListener {
    static OwlOntologyOutput newInstanceForOWL(OutputStream os, URI base);
}

class FoafOutput implements ISchematListener {
    static FoafOutput newInstanceForFOAF(OutputStream os, URI base);
}

class HigginsOutput implements ISchematListener {
    static HigginsOutput newInstanceForHiggins (OutputStream os,
                                                URI base);
}

class LdifOutput implements ISchematListener {
    static LdifOutput newInstanceForLDIF (OutputStream os,
                                         LdifOutputSettings settings);
}
```

The `LdifSchemaConverter` encapsulates adding an `OwlOntologyOutput` and `LdifInputSource` to an `ISchematInteractions`, to produce an OWL ontology of classes and properties which represent the schema obtained from the LDIF file.

```
class LdifSchemaConverter {
    static LdifSchemaConverter
        newInstanceForLDIFToOWL ( ISchematInteractions isi,
                                   InputStream is,
                                   OutputStream os,
                                   URI baseURI,
                                   Vector schemas);
}
```

The `SchematLocalScanner` provides scanning of a local file to obtain keywords. The `SchematRemoteScanner` performs a HTTP GET to obtain the requested file, and then uses `SchematLocalScanner` to parse it.

```
class SchematLocalScanner {
    static SchematScanResult parse ( ISchematInteractions isi,
                                     File localCopyFile,
                                     URI uri);
}

class SchematRemoteScanner {
    static SchematScanResult parse ( ISchematInteractions isi,
                                     File localCopyFile,
                                     URI uri,
                                     File tempDirectory);
}
```

The `SchematScanResult` class encapsulates the result of scanning a file. Some of the methods are listed below, others are used by the scutter for reparsing.

```
class SchematScanResult {

    Iterator iterateReferences();

    Throwable getException();

    void updateMap(Map m);
}
```

The Map updated by `updateMap` has String keys and values. The keys are:

Field key	Lucene Token Algorithm	Definition
local-name	single keyword	Local name of the file, constructed from the date the file was added and a unique identifier
uri	single keyword	The URI of the file
elements	space-separated set	The elements to be indexed, consisting of the URIs and semantic fragments from the ontology
last-parse-date	single keyword	Date of the last successful parse of the ontology
last-attempt-date	single keyword	Date of the last attempt by the scutter to request the ontology

last-modified-date	single keyword	Date the ontology was last modified
add-date	single keyword	Date this ontology was added to the index
dir-schema-dependencies	not stored in index	Names of RFCs containing schema used by this ontology
uri-dependencies	not stored in index	URIs of other ontologies on which this ontology is dependent
file-dependencies	not stored in index	Local names of other ontologies on which this ontology is dependent
sha1-hash	not stored in index	SHA-1 hash of the file
Format	single keyword	One of text/ldif, application/rdf+xml or application/owl+xml (the latter is not a valid MIME type)
Title, Creator, Subject, Description, Publisher, Contributor, Language, Relation, Coverage, Rights	indexed as text	from the Dublin Core specification [30]
Date, Type, Identifier, Source	single keyword	from the Dublin Core specification

The `SchematIndexer` class encapsulates performing updates to a Lucene search engine's on-disk database. The Map provided to `addDocument` contains index parameters obtained from `SchematScanResult` method `updateMap`.

```
class SchematIndexer {
    SchematIndexer(SchematIndexingConfig cfg);

    void addDocument(Map m);

    void deleteDocument(String fld, String txt);

    void deleteDocumentWithURI(String uristr);

    void close();
}
```

The `SchematSearcher` class encapsulates performing searches of the search engine's database. The elements key field index is searched by default.

```
class SchematSearcher {
    SchematSearcher(SchematIndexingConfig cfg);

    ISearchResultSet searchStringSortByDate(String s);

    void close();

    static ISearchResultSet getAllSortByDate(SchematIndexingConfig cfg);
}
```

A document store maintains a cache of ontology files and metadata. Each ontology's metadata, consisting of the keys used in the `updateMap` call (except for `elements`) is kept in its own file, encoded using a simple XML format.

The `SchematScutter` class uses `SchematRemoteScanner` and `SchematIndexer` to refresh the contents of the database. It also adds referenced ontologies to the database. In the current implementation, it only adds one level of reference for each scan run, rather than performing a 'depth-first' traversal.

```
class SchematScutter {
    SchematScutter (SchematIndexingConfig cfg);

    SchematScutterScanResult scan(Map pinglist);
}
```

Other classes to implement servlets are currently under development to support the Higgins and Identity Gang `identitieschemas.org` activity. This is the subject of a separate technical report.

## Availability

Snapshots of the Schemat software in a work-in-progress form can be downloaded from <http://www.ldap.com/>. It is available under a "BSD-style" license, the Informed Control Research Software License B.

Please note that as this software is intended for research, no optimizations have been incorporated, and performance has not been a goal of this project. Also this software has known bugs and limitations.

## Related Work

There have been several attempts to develop ontologies for directory service. Early in the deployment of enterprise directory services, meta-directory vendors proposed the specification of a 'white pages' person class for LDAP/X.500 directories, that would include all of the widely-deployed person attributes defined thus far.

The Service Provider Directory-enabled Network Applications (SP-DNA) effort [31], that was later integrated into the Directory Interoperability Forum (DIF), intended to define a model for interoperability between directory-enabled applications and directory services [32]. This model would capture semantics of a directory deployment not available in LDAP, but implemented in Service Provider applications. Secondly, it would allow a wide set of directory information trees to be conformant to SP-DNA, through a conceptual model that would be mapped onto a specific deployment's directory information tree.

Separately, Mr. Dietzold of the University of Leipzig Institute for Medical Information, Statistics and Epidemiology has defined a method for generating an OWL ontology from an LDAP schema, and RDF models for LDAP entries [33]. In that approach, attributes are turned into datatype properties, object classes into OWL classes, and the object classes of an entry determine the `rdf:type` of the entry.

## Potential Future Work

There are several areas of research that are intended to leverage the schemat tool. One is the construction of an Internet-accessible database of ontology mappings between well-known schemas. Another is the definition of new ontologies, for protocols such as Higgins or the WS-\* suite, that define the mappings between these protocols and existing "legacy" schemas such as inetOrgPerson.

In the initial implementation, a single non-RDF-based format (LDIF) and two RDF-based formats (FOAF, Higgins) were chosen for implementation. Subsequent schema parsing activity might be worthwhile for the output of the Schema Documentation Program for Windows 2000 [34], as well as XML-based encodings used in the SPML [35], DSML [36] or SAML [37] protocols.

Finally, it is anticipated that some of the algorithms used within schemat, once they have been refined and tested, will be extracted in order to provide a real-time ontology commentary and mapping function for identity schemas. This functionality will be particularly worthwhile for developing applications for emerging identity metasystems, such as that proposed around the Microsoft InfoCard format [38], in order to enable application developers to avoid being tied to a particular identity schema, and to be able to provide reasonable support in applications for community-defined schemas.

## References

- 1 S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, L. Stein, *OWL Web Ontology Language Reference*, February 2004, <http://www.w3.org/TR/owl-ref/>
- 2 G. Good, *The LDAP Data Interchange Format (LDIF) - Technical Specification*, June 2000, RFC 2849
- 3 K. Zeilenga, *Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map*, June 2006, RFC 4510
- 4 Microsoft Corporation, *Microsoft Identity Integration Server*, <http://www.microsoft.com/windowsserver/system/miis2003/default.aspx>
- 5 Sun Microsystems Inc., *Meta-Directory Deployment Guide 5.0 SP1*, <http://docs.sun.com/app/docs/doc/816-6098-10>
- 6 Novell, Inc., *DirXML*, <http://www.novell.com/products/dirxml/>
- 7 J. Clark, *XSL Transformations (XSLT)*, November 1999, <http://www.w3.org/TR/xslt>
- 8 L. Kennard, *Check Out That DirXML Engine*, May 2000, [http://support.novell.com/techcenter/articles/nc2000\\_05b.html](http://support.novell.com/techcenter/articles/nc2000_05b.html)

- 9 Sun Microsystems Inc., *Sun ONE Integration Server EAI Edition 3.1*, [http://www.sun.com/software/products/integration\\_srvr\\_eai/home\\_int\\_eai.xml](http://www.sun.com/software/products/integration_srvr_eai/home_int_eai.xml)
- 10 M. Wahl, *Organizing principles for identity systems: decentralized l10n*, February 2005, [http://www.ldap.com/1/commentary/wahl/20050203\\_01.shtml](http://www.ldap.com/1/commentary/wahl/20050203_01.shtml)
- 11 D. Brickley, L. Miller, *FOAF Vocabulary Specification*, January 2006, <http://xmlns.com/foaf/0.1/>
- 12 M. Wahl, *P3P Policy Attributes for LDAP*, February 2005, <http://www.ldap.com/1/spec/schema/p3p-spec.html>
- 13 F. Manola, E. Miller, *RDF Primer*, February 2004, <http://www.w3.org/TR/rdf-primer/>
- 14 Eclipse Foundation, *Higgins Wiki*, [http://wiki.eclipse.org/index.php/Higgins\\_Wiki](http://wiki.eclipse.org/index.php/Higgins_Wiki)
- 15 T. Berners-Lee, R. Fielding, L. Masinter, *Uniform Resource Identifier (URI): Generic Syntax*, January 2005, RFC 3986
- 16 D. Recordon, B. Fitzpatrick, *OpenID Authentication*, May 2006, [http://openid.net/specs/openid-authentication-1\\_1.html](http://openid.net/specs/openid-authentication-1_1.html)
- 17 T. Wason, *Liberty ID-FF Architecture Overview*, <http://www.project-liberty.org/specs/draft-liberty-idff-arch-overview-1.2-errata-v1.0.pdf>
- 18 M. Biddulph, *Crawling the Semantic Web*, 2004, [http://www.idealliance.org/papers/dx\\_xmle04/papers/03-06-03/03-06-03.html](http://www.idealliance.org/papers/dx_xmle04/papers/03-06-03/03-06-03.html)
- 19 M. Smith, *Definition of an X.500 Attribute Type and an Object Class to Hold Uniform Resource Identifiers (URIs)*, January 1997, RFC 2079
- 20 M. Smith, *Definition of the inetOrgPerson LDAP Object Class*, April 2000, RFC 2798
- 21 K. Zeilenga, *Lightweight Directory Access Protocol (LDAP): Directory Information Models*, June 2006, RFC 4512
- 22 A. Sciberras, *Lightweight Directory Access Protocol (LDAP): Schema for User Applications*, June 2006, RFC 4519
- 23 K. Zeilenga, *COSINE LDAP/X.500 Schema*, June 2006, RFC 4524
- 24 J. Carroll, *ARP: Another RDF Parser*, <http://www.hpl.hp.com/personal/jjc/arp/>
- 25 S. Bechhofer, P. Lord, R. Volz. *Cooking the Semantic Web with the OWL API*. 2nd International Semantic Web Conference, ISWC, Sanibel Island, Florida, October 2003. (software at <http://sourceforge.net/projects/owlapi>)

- 26 E. Tsai, *Netscape Directory SDK for Java: Source Code Release*, <http://www.mozilla.org/directory/javasdk.html>
- 27 Apache Software Foundation, *Apache Lucene - Overview*, <http://lucene.apache.org/java/docs/index.html>
- 28 Apache Software Foundation, *HttpComponents Overview*, <http://jakarta.apache.org/httpcomponents/>
- 29 Sun Microsystems Inc., *Java Servlet Technology*, <http://java.sun.com/products/servlet/>
- 30 D. Beckett, E. Miller, D. Brickley, *Expressing Simple Dublin Core in RDF/XML*, July 2002, <http://www.dublincore.org/documents/dcmes-xml/>
- 31 Nortel Networks, *SP-DNA - Service Provider Directory-enabled Network Applications Homepage*, August 2001, <http://web.archive.org/web/20010802144513/standards.nortelnetworks.com/dif-sp-dna/index.html>
- 32 M. Wahl, *Organizing principles for identity systems: Background: SP-DNA metaschema*, June 2005, [http://www.ldap.com/1/commentary/wahl/20050614\\_01.shtml](http://www.ldap.com/1/commentary/wahl/20050614_01.shtml)
- 33 S. Dietzold, *Generating RDF Models from LDAP Directories*, [http://www.semanticscripting.org/SFSW2005/papers/Dietzold-RDF\\_Models\\_from\\_LDAP.pdf](http://www.semanticscripting.org/SFSW2005/papers/Dietzold-RDF_Models_from_LDAP.pdf)
- 34 Microsoft Corporation, *Schema Documentation Program*, <http://www.microsoft.com/technet/prodtechnol/windows2000serv/technologies/activedirectory/maintain/schema.mspx>
- 35 OASIS Open, *OASIS Service Provisioning Markup Language (SPML) Version 2*, <http://www.oasis-open.org/committees/download.php/17708/pstc-spml-2.0-os.zip>
- 36 OASIS Open, *Directory Services Markup Language (DSML) v2.0*, <http://www.oasis-open.org/committees/dsml/docs/DSMLv2.xsd>
- 37 OASIS Open, *Security Assertion Markup Language (SAML) v2.0*, <http://docs.oasis-open.org/security/saml/v2.0/saml-2.0-os.zip>
- 38 D. Chappell, *Introducing Windows CardSpace*, April 2006, <http://msdn.microsoft.com/windowsvista/default.aspx?pull=/library/en-us/dnlong/html/IntroInfoCard.asp>