A Lightweight Ontology Repository
Jin Pan, Stephen Cranefield and Daniel Carter
University of Otago
Department of Information Science
PO Box 56, Dunedin, New Zealand
panji016@infoscience.otago.ac.nz

ABSTRACT
Multi-agent systems rely on shared ontologies to enable unambiguous communication between agents. An ontology defines the terms or vocabularies used within messages encoded using an agent communication language (ACL). In order for ontologies to be shared and reused, ontology repositories are needed to store and maintain ontologies. The Otago Ontology Repository is a prototype implementation of a design to allow ontology designers and agents to use open Web standards to publish and retrieve ontologies and metadata about them. Key features of the system include the use of the HTTP protocol following the REST architectural style, the representation of the recorded information about ontologies and the repository information schema using RDF and its schema language RDFS, and the use of URNs to identify ontologies. The use of Web standards for communication between agents and Web-based resources such as ontology repositories enables a more lightweight and open architecture for agent interaction with these resources.

Categories and Subject Descriptors
I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence—Multiagent systems.
H.3.5 [Information storage and retrieval]: Online Information Services—Web–based services.

General Terms
Design.

Keywords
Multi-agent systems; ontology repository.

1. INTRODUCTION
In a multi-agent system, ontologies are important for providing a shared understanding of domain knowledge to allow agents to communicate in an open interoperable environment. An ontology is “an explicit specification of the structure of a certain domain and includes a vocabulary for referring to the subject area, and a set of logical statements expressing the constraints existing in the domain and restricting the interpretation of the vocabulary” [11]. An ontology provides a mechanism to enable agents to have shared understanding of the semantics for terms used in agent messages. When ontologies are shared between sending agents and receiving agents, messages can be interpreted correctly by the receiving agents, thus reducing the possibility of misunderstanding. However, in order for ontologies to be shared and reused, ontologies have to be published in repositories for later reference by system designers and agent applications.

Ontology repositories have previously been developed for enterprise systems and the Semantic Web, including DOME [28] and ONTOSERVER [26]. Ontolingua [8] is a Web-based collaborative environment for creating, browsing and editing ontologies. FIPA specified an ontology service for agent communication in 19981 [11] and there exists at least one implementation [22]. The FIPA ontology service uses the Open Knowledge Base Connectivity (OKBC) protocol [3] as a bridge to ontology repositories, and an Ontology Agent (OA) as a component that mediates between other agents and ontology servers.

We believe that for agent and ontology technology to gain widespread acceptance, it should be possible for agents to directly access ontologies using a well-known protocol, and the information schema underlying an ontology repository should be easily accessible in a declarative format. Therefore our approach differs from the ontology service proposed by FIPA, which requires the existence of a specialised ontology agent that mediates between other agents and knowledge representation systems storing ontologies, and the OKBC protocol, which specifies a procedural interface for querying knowledge representation systems. Instead we base our repository architecture on the Web standards HTTP and RDF. Also, as repositories may be moved from host to host, and/or replicated in different locations, we chose the Web standard for location-independent identifiers—Uniform Resource Names (URNs)—to identify ontologies in the repository.

There are a number of facilities that an ontology server may provide:

- support for creating and editing ontologies,

1 FIPA has since classified this specification as obsolete pending the outcome of a new FIPA technical committee focusing on ontology standards for multi-agent systems.
• support for publishing and retrieving ontologies (by humans via a graphical user interface and/or via a network protocol),
• support for recording metadata about ontologies and the relationships between them,
• support for interactively browsing the structure of ontologies, and
• inference mechanisms to verify the consistency of ontologies, etc.

The repository described in this paper is designed to be “lightweight”, focusing on the publishing and retrieval of ontologies and metadata about them. As the repository may contain ontologies expressed in different modelling languages, it is left to external tools to provide editing capabilities. However, future extensions of the repository could allow queries and inferences about the structure of ontologies to be made (by extending the RDF meta-ontology used).

The structure of the paper is as follows. Section 2 describes the FIPA specification for ontology services. Section 3 discusses the design of our lightweight ontology repository. Section 4 describes the details of the implementation. Section 5 illustrates how human users and agents interact with the repository. Section 6 concludes the paper with a discussion of the design decisions and benefits of our approach and the limitations of the current implementation.

2. FIPA SPECIFICATION FOR ACCESS TO ONTOLOGY SERVICES

The FIPA ontology service reference model defines an Ontology Agent (OA) that wraps and hides the back-end repository implementation, leaving the design and implementation of this to developers. The OA provides ontology services for other agents (such as query and update), which use ACL messages and the FIPA ontology service meta-ontology to communicate with the OA.

The OKBC interface and language bindings are used by the OA to access heterogeneous ontology repositories. OKBC is a set of generic operations for connecting user agents with knowledge bases stored in knowledge representation systems (KRSs). This interface isolates an application from the implementation details of different KRSs and enables the development of generic tools (e.g. graphical browsers and frame editors) that operate on KRSs [3]. The problem with OKBC from the FIPA perspective is that its client-server model API does not fit the FIPA style of agent communication. Therefore, the OA is used to encapsulate the use of OKBC so other agents do not need to implement it.

The FIPA ontology service specification defines a meta-ontology for conversations between FIPA agents and the OA about ontologies. This model, based on the OKBC knowledge model, declares a set of actions and predicates that can be used in the content of the conversation with the OA. The mapping between these FIPA ACL messages and the OKBC API is performed by the OA.

The OA may be a sophisticated ontology server capable of mapping/matching between ontologies but the specification states that implementations do not have to support all the specified features, indeed it is expected that not all ontology agents will support the full specification due to its complexity.

Figure 1 shows the FIPA ontology service reference model which depicts the interaction among agents, the OA, and the back-end ontology repositories.

3. LIGHTWEIGHT ONTOLOGY REPOSITORY

This section discusses the key features of our repository design. The Representational State Transfer architectural style (REST) is used as the fundamental architecture of our repository. A meta-ontology is designed to model the structure of ontologies stored in the repository. The rdftp server [25] stores the ontology metadata and allows both humans and agents to query it. URNs are used as ontology identifiers.

3.1 The REST Architectural Style

The Representational State Transfer architectural style (REST) was designed as an abstract model of the principles underlying the World Wide Web architecture in general and the HTTP protocol in particular. REST aims to “minimize latency and network communication, while at the same time maximizing the independence and scalability of component implementations” [9]. In particular, REST systems are considered to be highly scalable due to the use of stateless communication and their support for the caching of request responses (response messages must indicate whether they can be cached). Two of the key principles underlying REST are the following:

- The notion of a ‘resource’, identified by a resource identifier, as the fundamental unit of information. Resources are abstract notions that are not directly transferred across a network. Instead a representation of the current or intended state of a resource is transferred between system components. This consists of a sequence of bytes together with metadata describing those bytes. Multiple representations of a resource may be available using different media types [19].
- The constraint that there should be a uniform interface for interactions between system components. This can be summed up by the following assertion: “no matter what your
problem, you can and should think about it as a data resource manipulation problem rather than as an API design problem" [20]. When applied to HTTP-based communications on the Web, this means choosing to make use of the existing HTTP request types such as GET, POST and PUT in ways that respect their semantics (e.g. POST creates a new resource and GET retrieves one) rather than encoding component-specific operations by including a query expression at the end of an HTTP URL.

Following the REST philosophy, ontologies in our repository are considered to be abstract resources that are identified by World Wide Web Uniform Resource Identifiers (in fact Uniform Resource Names, URNs [17] are used), and the repository stores possibly multiple representations of each ontology using different media types (e.g. image/jpeg, application/rdf+xml [24] or application/x-xmi1.0-uml1.3+xml2). The REST principles are applied to the repository by using an HTTP POST request to create a new ontology resource in the repository, with a representation of the ontology in some specified media type included in the body of the request. A URN for the new resource is returned in the response message using the Location header (a non-standard header 'URNHint' is used in the POST message if the client wishes to influence the repository's choice for the generated URN). Alternative representations of the ontology in different media types can subsequently be uploaded using HTTP PUT. Ontologies are downloaded by performing an HTTP GET, with the Accept header specifying the desired media type (the URN must first be resolved to a URL and the mechanism used for this is described in Section 3.4).

3.2 The Repository Meta-Ontology

An ontology repository must have some underlying model of the structure of the information that is stored in it. Depending on the design decisions taken, this could range from the schema for the underlying database to a full meta-level ontology that describes the concepts used to represent information about ontologies. We have taken the latter approach, using the Resource Description Framework Schema (RDFS [15]).

Figure 2 shows the meta-ontology for our repository. Instead of showing the RDF schema itself, for clarity and brevity we use a profile of the Unified Modeling Language that has been developed for ontology modeling [4]. A corresponding RDF schema can be generated automatically from this UML class diagram [5]. The stereotype «resourceType» indicates that objects of these classes are resources (in the sense defined by REST), and the tagged values `{refType = URN}` indicate that the references used to identify the objects are Uniform Resource Names. The stereotype «valueType» indicates a class that is not being treated as a resource, i.e. it will not be identified by any type of reference.

Each ontology may be based on an abstract conceptualisation which has version and name properties:

“A conceptualisation is an abstract, simplified view of the world that we wish to represent for some purpose. Every knowledge base, knowledge-based system, or knowledge-level agent is committed to some conceptualisation, explicitly or implicitly. An ontology is an explicit specification of a conceptualisation.” [12]

A title, version number, language, and textual description are used to describe an ontology. For example, a Family ontology may be written in the language UML, and its version might be 1.0. Each ontology can have one or more authors, represented by the Person class. The Representation class represents a particular encoding of an ontology, person or conceptualisation. For example, if the language of an ontology is RDF, an ontology can be encoded using any of the formats RDF/XML, N-triples or N3. A person resource might be represented by a GIF file and a text file can represent a conceptualisation. The associations between the Ontology, Person and Conceptualisation classes and the Representation class make explicit the mapping between resources and representations that must be maintained by a component of a REST system.

At present this meta-ontology is rather simple, addressing only information about ontologies as a whole. It would be straightforward to extend this to include information about concepts defined within ontologies and the relationships between them (once an appropriate set of relationships has been chosen).

3.3 Rdftp and Ontology Metadata

The repository meta-ontology is used for ontology developers to assert metadata about the ontology and can be used by clients (both humans and agents) to express queries about the properties of a particular ontology, or to locate an ontology having particular properties. For example, developers might want to publish information about the author or the version of the ontology to the repository. To allow clients to assert and query information about ontologies, the rdftp protocol is used [25]. This provides an HTTP encoding of RDF triple assertions and queries.

2 This unregistered media type uses a modification of the scheme proposed by Curt [6].
Rdftp is a RDF server that supports queries and updates of RDF resources. The rdftp server stores RDF content both in a relational database as RDF triples and in a hierarchical repository of RDF models, residing in the server’s file system as RDF files. The rdftp server is implemented as a PHP script and runs on top of any HTTP server that supports PHP. Both rdftp query and update operations can be called using either the HTTP GET or the HTTP POST method. Rdftp also provides a simple HTML interface for “Navigating” through the query results. In our repository, the rdftp server is used to store the ontology metadata and the URN to URL mappings described in Section 3.4.

3.4 URNs as Ontology Identifiers

For ontologies to be reusable and shareable, there needs to be a standard mechanism for identifying and accessing an ontology. As the World Wide Web infrastructure is now ubiquitous, the Web notion of identifier, Uniform Resource Identifier (URI), seems a sensible choice. However, with the prevalent form of URI—Uniform Resource Locators (URLs) using the http scheme—there is a major shortcoming for the purpose of identifying and retrieving ontologies. An http URL directly refers to a particular host from which a representation of the resource may be obtained. This means that an ontology identified by a URL could not be made available at multiple hosts (introducing problems of scalability) or there might be multiple URLs referring to the same ontology (making it difficult for agents to establish that they are using a common ontology).

There is another form of URI that provides exactly the characteristics required for ontology references—“location independent identification of a resource, as well as longevity of reference” [7]—namely the Uniform Resource Name (URN). FIPA has lodged an application for a URN namespace for all its identifiers (including ontologies) [2] and the Agentcities project [27] is using an unofficial URN scheme for identifiers [1]. However, accessing ontologies using URNs requires additional network infrastructure that is not part of current mainstream Web technology, for example the resolution of URNs to location-dependent URLs that can then be used to retrieve one particular copy of the resource. In order to allow ontologies in our repository to be identified by URNs, we have provided this infrastructure in the following way. When an ontology is posted to the repository, it is made accessible via a particular URL on the repository server’s host. The URN that is generated for the ontology is then associated with the URL by using rdftp to assert a triple representing the mapping. In this implementation, the ‘tag’ URN scheme [14] has been used as this is independent of any particular resource-location or identifier-resolution scheme. To access an ontology using a URN, an agent (or a GUI client for human users) must first use rdftp to look up the currently associated URL. If the repository is moved to a different host, the RDF triple representing the mapping can be changed. This simple approach is sufficient for the present, but a more sophisticated mechanism involving a hierarchy of mapping databases will be required in the future to allow for replicated repositories. There have been some recent promising developments in this area [16].

4. IMPLEMENTATION

Our repository allows system designers and agents to update and query ontologies using the standard HTTP protocol and RDF. A MySqI database is used to store ontologies. HTTP stands for the Hypertext Transfer Protocol. It is the network protocol used to access resources on the World Wide Web. HTTP uses the client-server model: an HTTP client opens a connection and sends a request message to an HTTP server; the server then returns a response message. After delivering the response, the server closes the connection.

Java servlets are implemented to handle HTTP requests to the repository by accessing the ontology database and rdftp server. System designers and agents may also directly access the rdftp server using the HTTP protocol as clients. When a client publishes an ontology, it uses HTTP POST, and the HTTP Content-Type header is used to specify the media type of the ontology representation (e.g. application/ rdf+xml). The client includes the ontology as the HTTP POST body. The repository servlet stores the representation as a Binary Large Object (BLOB) in the ontology database along with its media type, and returns a new URN identifying this ontology. Figure 4 shows the process of publishing an ontology in the repository using HTTP POST. The final arrow illustrates how the returned URN can subsequently be used to record additional information about the ontology resource using rdftp.

When an ontology resource is created in the repository as the result of a POST message, it is assigned a URL as well as a URN. The ontology representation is stored in the repository database, indexed by its media type and the path component of the URL. The generated URN is also stored, indexed by the URL. The URN to URL mapping is asserted to the rdftp server (using an RDF property defined specifically for this purpose). Clients wishing to access an ontology using a URN must first use the
rdftp protocol to look up this mapping (which may be one-to-many if ontologies are stored in multiple repositories), and can subsequently choose one of the resulting URLs to access the repository server.

This design provides a simple mechanism for supporting the use of URNs as ontology identifiers, and the consequent benefits include the ability to move a repository to a different host, and the possibility of replicating a repository. The mapping mechanism is currently reliant on the use of a single rdftp server and will therefore not scale well, but this is seen as an interim measure until the standard Web infrastructure provides a better mechanism [16].

The rdftp server has a second role: it is used to store metadata about resources in the repository (currently these resources can be conceptualisations, ontologies and authors). This information is published there by clients using HTTP POST messages according to the rdftp specification.

To provide a convenient way for agents to resolve URNs using the rdftp server we have implemented this functionality as a Java Agent Services (JAS) [13] ‘service’ in our FIPA and JAS agent platform Opal [21]. Figure 5 shows how an Opal agent can query the ontology metadata to find an ontology of interest, resolve the URN and then retrieve the ontology. Once the URN is resolved to a URL, the agent issues an HTTP GET message for that URL, and specifies the media type using the HTTP Content-Type header. In the repository implementation, a single servlet handles requests for all URLs associated with resources in the repository. This retrieves the ontology BLOB from the ontology database, and returns it to the agent. The HTTP response message also includes a Location header specifying the URN that should be used as the permanent, location-independent identifier for that resource.

5. EXAMPLE USAGE

The repository supports both human and agent access. The screenshot in Figure 6 shows part of the graphical user interface for access to the ontology repository.

This form allows humans to publish ontology metadata (based on the repository meta-ontology) to the repository. Users also have flexibility to upload a new ontology representation and create authors before posting the ontology metadata. The user interface uses HTTP POST and GET requests and the rdftp protocol to interact with the ontology repository.

Figure 7 shows an example scenario in which an Opal agent accesses the ontology repository to interpret a message from another agent.

Agent 2 receives a message from agent 1. The message refers to an unknown ontology URN in the ontology element of the message. Agent 2 needs to refer to the specified ontology to interpret the terms used inside the message content. In particular, we assume that agent 2 has the ability to generate Java classes automatically from an ontology defined using UML (by applying

Figure 4: Publishing an ontology using HTTP POST

Figure 5: Locating an ontology by querying the repository

Figure 6: A screenshot from the client applet

Figure 7: Scenario diagram implemented in the Opal platform
an XSLT stylesheet to the UML model serialized in the XMI format) [4]. If it can ascertain that the ontology is defined in UML and an XMI representation is available, then it can use this mechanism to internalise the received message as a network of Java objects—a convenient form for its subsequent processing.

Agent 2 begins its interaction with the ontology repository by querying the metadata in the rdftp server to find out the language that the ontology is written in. The response confirms that the language is UML. Agent 2 then uses the Opal platform’s URN resolution service to find a URL corresponding to the desired ontology’s URN. It then sends an HTTP GET request to the ontology repository using the Accept header to indicate its desire to receive an XMI representation, and receives the ontology in the XMI format in the response message. Agent 2 then applies its XSLT stylesheet to produce corresponding Java classes, loads them and continues its processing of the message from agent 1.

In this scenario, if agent 2 needs to make a deeper analysis of the ontology, it could use the Java Metadata Interface (JMI) [23] or the Novosoft UML library (NSUML) [18] to load the XMI file and then explore the UML model’s structure using Java method calls.

6. DISCUSSION
The design of our ontology repository is based on the view that industry uptake of agent technology will be more likely if it can be based (where possible) on widely known Web standards, rather than specialised protocols such as OKBC. We also feel that it is unnecessary to require agents to interact with an ontology repository using an ACL—by applying the REST architectural style we have found that the HTTP POST, PUT and GET operations provide a suitable model for publishing and retrieving ontologies. One consequence of our use of HTTP is that there seems to be no need for a centralised ontology agent, in contrast to the FIPA ontology service. The purpose of FIPA’s OA is to translate between ontology service requests expressed using FIPA ACL and the equivalent operation in OKBC—it was clearly assumed that it would not be reasonable to expect agent developers to include OKBC querying abilities in their agents. In contrast, we believe that it is reasonable to expect agent developers to implement HTTP GET requests in their agents given the common need for this type of programming in modern application development and the good code library support available for HTTP client programming. However, HTTP has many features that aren’t commonly needed in simple Web programming, and as our repository becomes more fully featured and makes greater use of the available HTTP response codes and mechanisms for authentication, etc., it will be necessary to re-evaluate this claim. Even if there is a higher demand than initially expected on the HTTP knowledge of agent developers, this may be an acceptable trade-off for the elimination of the OA as a potential bottleneck and single point of failure. The use of URNs to identify ontologies also allows greater scalability of our ontology support infrastructure by replication of the repository and the association of multiple URLs with each resource URN (although there is currently a lack of a scalable URN resolution service).

Another key design decision is the use of RDF to represent ontology metadata in terms of a repository meta-ontology defined using RDF Schema. This meta-ontology provides a public declarative view of the information structure of the repository that can be examined by both agents and humans using the repository, in order to ascertain the type of assertions and queries that can be made. It also allows the repository to be easily extended to include additional types of information, such as the internal structure of ontologies (by modelling concepts as resources and defining RDF properties for expressing metadata about them and the relationships between them).

Our repository is designed to support the use of multiple ontology languages and encodings. Therefore we have focused solely on storage, query and retrieval functionality and we leave the creation and editing of ontologies, and other language-specific ontology manipulations and analyses to external tools. However, we envisage that a plug-in architecture could be developed to allow such tools to be integrated with the repository in the future. The current implementation is a prototype that lacks some crucial features, such as support for the authentication of users and the deletion of resources. It is intended to add these features using existing mechanisms in HTTP.

7. ACKNOWLEDGEMENTS
This research was funded by a University of Otago research grant.

8. REFERENCES


