Knowledge Management Using Semantic Web Languages and Technologies

Steven Gollery
Collaborative Agent Design Research Center (CADRC)
Cal Poly, San Luis Obispo, California
6/30/2004

1. Executive Overview

Each organization of more than a few persons generates a number of documents containing information about the activities of the organization. The sum of these documents provides a kind of organizational memory that can be used both in evaluations of past performance and as the basis of future planning based on previous experience.

Unfortunately, the nature of these documents mitigates against the reuse of information for purposes other than those for which each document was originally intended. To locate all the information about some aspect of the organization, it frequently becomes necessary for human beings to search a mass of documents to manually extract and collate all text related to the problem at hand. It would be advantageous if a software system could be used to find all relevant information and assemble it into a single document for input to the planning or evaluation process. But the current document-centric approach to information organization limits the functionality and utility of such a software system.

The Semantic Web Knowledge Management research project being undertaken by the Collaborative Agent Design Research Center (CADRC) at the California Polytechnic State University (Cal Poly), San Luis Obispo postulates that a more useful organizational memory can be created by separating the information from the structure imposed when that information is embedded in specific documents. We propose a system that allows users to create informational units rather than documents. Document structure is specified by objects outside the units themselves, enabling the reuse of information in multiple documents while maintaining the referential integrity of each organizational unit.

Project researchers in the CADRC Center are working towards the production of a test-bed system to explore various approaches to information entry and retrieval. To support such a system, they have defined a preliminary information structure known as an ontology using the semantic web language OWL (OWL 2004, Antoniou and Harmelen 2004). Tests on the use of this ontology as the basis of the system are ongoing.

This report of research completed under C3RP funding during FY04-05 discusses the information needs and representational approach for a knowledge management system using semantic web technologies.

2. Introduction

The Semantic Web Knowledge Management project is intended to demonstrate the use of semantic web languages, concepts and technologies in the design and implementation of a knowledge repository for the support of the decision-making process. Initially, we are focusing on concepts and functionality related to an organization involved in software development, using the kinds of information that are generated in the course of development projects in our Center at Cal Poly as a test case. However, we expect that the ontology and supporting tools are relevant to other organizations as well.

During the software development process, large amounts of information and knowledge are recorded in individual documents. Collectively, these documents form an organizational memory that can be used as part of making decisions for individual projects and for the entire organization. The knowledge management project explores the potential value of storing information in a single common repository in a form that can support the creation of tools that provide access based on an explicit representation of the semantics of each individual piece of information.

The remainder of this report is a discussion of the different sections of the ontology and what sort of functionality each section is intended to support. The ontology itself is divided into two parts. One part contains concepts and relationships that can be used to describe an organization and its projects. This part is fairly straightforward. The other area covered by the ontology represents the information itself, from several points of view. These two areas are covered individually. Following the knowledge management section, the report covers the association of topics with various parts of the knowledge management ontology, and the use of external ontologies.

As the two parts of the ontology have evolved, it has become apparent that there is ambiguity over which part should contain a given concept. For instance, it seems fairly clear that a project team is part of an organization. Each team is responsible for one or more projects, and each project has a list of requirements. Each requirement is a piece of knowledge about the project. The question then becomes: Are requirements part of the organization ontology, or are they part of the knowledge management ontology?

Currently, requirements are modeled as part of the organization because of their close association with a project. But it would be equally reasonable to include requirements in the knowledge management area. This and similar questions are likely to be revisited in the future.

3. The Organization Ontology

The Organization ontology (Figure 1) represents concepts and relationships for describing organizations. This is currently a minimalist view: it includes just enough to allow the definition of the part of an organization the deals with Projects, Requirements, Tasks, and other related information. A complete ontology for organizations would naturally be much larger. Note: in the rest of this document, where it is necessary for elements of the Organization ontology to be distinguished from elements of other ontologies, the Organization elements will be identified using the prefix “ORG.”

The Organization ontology uses parts of several ontologies designed by other sources. There is more about the use of outside ontologies later in this report. One of the most important external contributions for the organization ontology is “Friend of a Friend”, or FOAF. FOAF is a very popular ontology that is referenced in many other sources. We use it here as the basis for many of our people-related classes and properties.

The Agent class is the base for several important classes. It refers to anything that can act, and so includes both Person and Organization. ORG:Agent inherits from FOAF:Agent, and adds properties such as “is responsible for” (zero or more Tasks), “is involved with” (zero or more Projects), and “administers” (zero or more Projects).

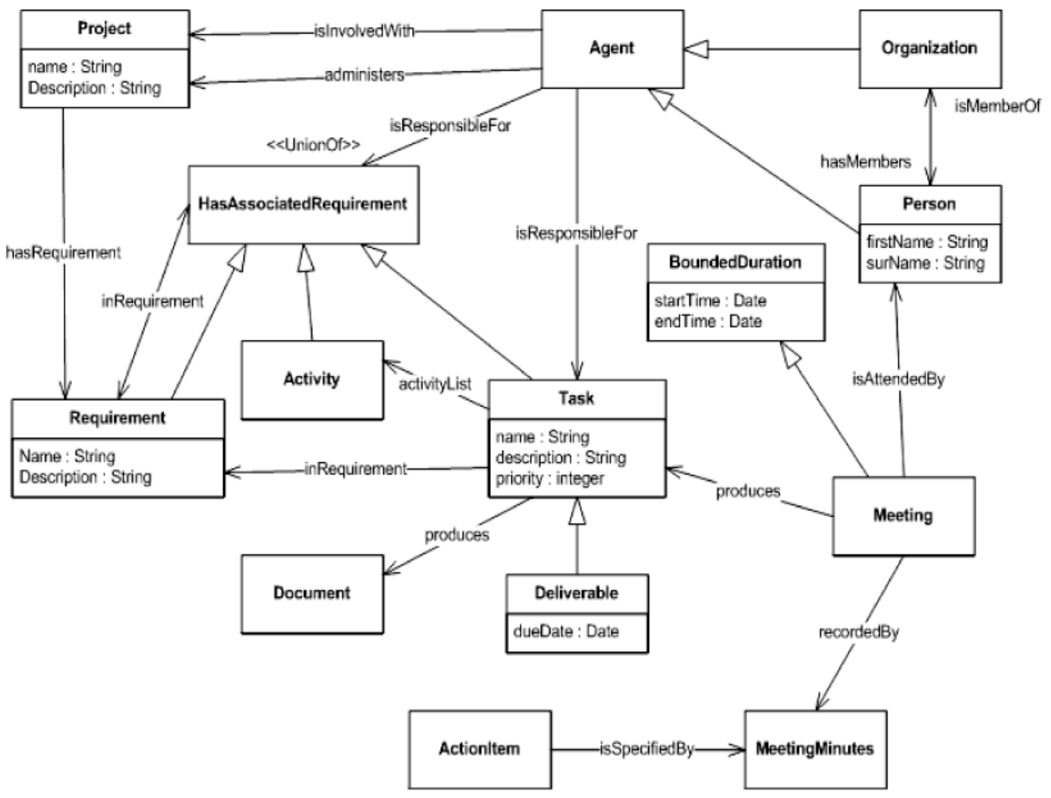

Figure 1: Classes and properties for describing an organization (partial)

Person is a subclass of both ORG:Agent and FOAF:Person (which is itself a subclass of FOAF:Agent, among others). To the properties inherited from Agent, the Person class adds a first name and a surname, and an “attends” property whose value is zero or more instances of Meeting. A Person is also a member of zero or more Organizations.

Organization is a subclass of ORG:Agent and of FOAF:Organization. An Organization has a name and a list of members (instances of Agent).

Project is a subclass of FOAF:Project. Each project has a name, and may have a description, a sponsor, and a list of Agents that are involved in the project. Projects also potentially have an “is described by” property whose value is an instance of Document, and may have a DecisionDocument associated with it. However, these last two are subject to review: it may be that these concepts are covered by classes and relationships in the knowledge management ontology (Figure 2).

A Project also has a list of Requirements. This is currently an unordered list. A Project may have an “is bound by” property whose value is an instance of Contract.

BoundedDuration is the base class for all classes that have a start time and an end time.

Meeting is a subclass of BoundedDuration. Each instance of Meeting “is attended by” a list of Person objects and “produces” zero or more Tasks. It is likely that Meeting will be extended further, either here or in the Knowledge Management ontology. The intent of Meeting is to provide a target for the MeetingMinutes document type, so that minutes can be entered into the system as fragments, to later be assembled for presentation, either stand-alone or as part of a decision document.

Requirement is a subclass of BoundedDuration. A Requirement has a name and a description, and a lists of Activities, Tasks, and other Requirements. This list is ordered, since some of the elements will have dependencies. Each Requirement is associated with exactly one Project.

Task is a subclass of BoundedDuration. A Task is “defined at” a Meeting, “produces” a Document, “is bound to” a Contract, has a description, a date issued, a priority, and a name. Each Task also has a list of Requirements of which the task is a part. Additionally, a Task may have an ordered list of Activities directly defined for it. Finally, a Task may of course also inherit Activities from the Requirements to which it belongs.

An Activity has a name and a description. Activities apply to all tasks that are defined within the same Requirement, and are inherited by sub-Requirements.

ActionItem “is specified by” MeetingMinutes. This class is currently under evaluation while we are investigating whether it is needed? If so, what does it need as properties?

4. The Knowledge Management Ontology

The key assumption of the knowledge management (*km*) ontology (Figure 2) is that the information in a document is more fundamental than the document itself, and that storing information separately from the document structure will allow information to be used in multiple contexts. To that end, the knowledge management ontology is divided into three areas: Fragments; Assemblies; and, DocumentTypes. Each of these areas is discussed below.

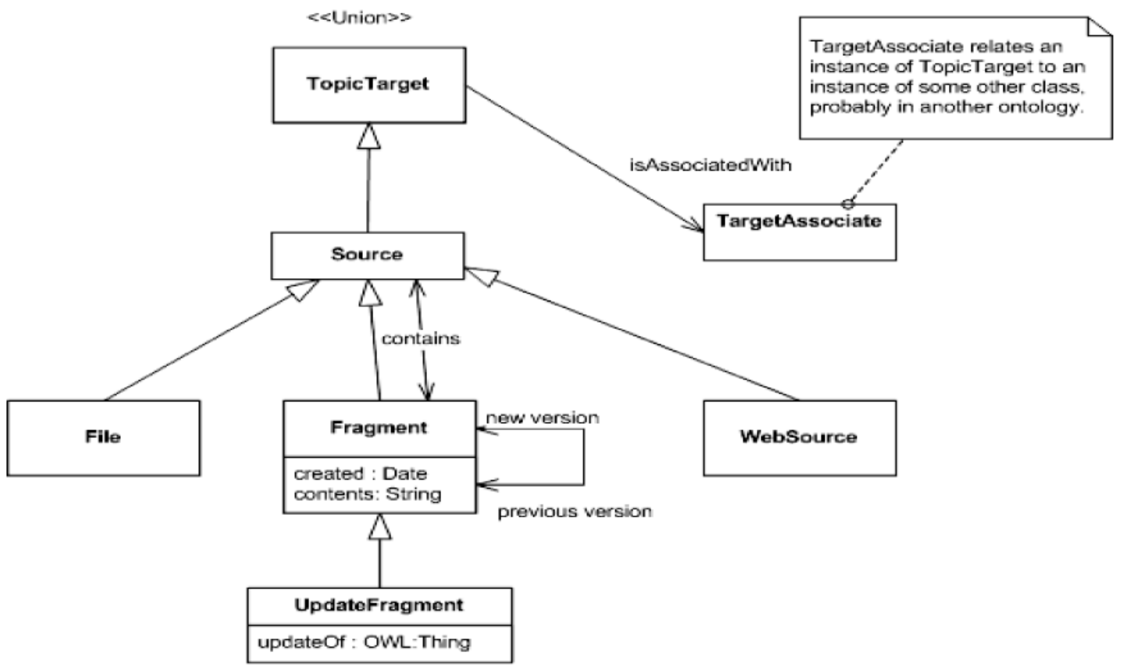

Figure 2: Partial ontology for representing textual information units.

4.1 Fragments

Fragments are the most fundamental elements of the ontology. Each fragment is intended to contain a reasonably self-sufficient piece of knowledge or information. For instance, a report on the current status of a requirement could be considered as a single fragment.

The structure of a Fragment is very simple. Each Fragment has a “contents” property whose value is a string of arbitrary length. Each Fragment also has a “created” property whose value is a Date. This allows systems to organize fragments chronologically for presentation if a user has that need.

Each Fragment also has a “previous version” and a “next version” property that point to other instances of Fragment. This provides for a simplistic form of version control.

Currently, Fragment is a subclass of Source. Each instance of Source may contain multiple Fragments, each of which in turn may be contained by many instances of Source. It should be noted that this was an early idea of how to handle the concepts now fleshed out more fully in the Assembly area (below). It is unknown whether it will be useful to maintain the “contains” relationship in the future.

Fragment also has one subclass, UpdateFragment, which allows users to enter some new information relating to another object. This might allow, for instance, an update to some information originally gleaned from knowledge acquisition and therefore differs from “new version” because an update is an addition to existing information, while a new version replaces the old version. Also, an UpdateFragment can update anything, while the “version” properties always associate two Fragment instances.

The Fragment concept is very simple, but it is expected that the bulk of the knowledge repository will be made up of Fragment instances.

In addition to Fragment, Source has two other subclasses: File; and, WebSource. These exist for two purposes. First, to allow users to associate topics with resources that are external to the repository structure, and second, to enable Reference objects (see below) to refer to external resources, as in citing another document.

4.2 Assemblies

One of the uses of Fragments is to provide the contents of one or more documents. To accomplish this, the ontology defines the concept of an Assembly. Each Assembly structures a set of Fragments into a single document. An Assembly is a generic means of representing the organization of information for some specific purpose (Figure 3).

Assemblies provide the ability to define a specific order for a collection of Fragments. Each Assembly includes an ordered list of Part instances. Each Part is associated with a Fragment or some other type of object that can be included in a visual presentation. A document, then, is a set of Fragments organized according to an Assembly.

Each Part instance is contained by one or more Assemblies. Part currently has four subclasses: FragmentPart; Reference; ReferenceList; and, ObjectInsert. This is not intended as a comprehensive list of all potential types of document sections, but represents simply a convenient starting point.

A FragmentPart “refers to” an instance of Fragment. This is likely to be the most often-used class in an Assembly. When a FragmentPart is encountered in the “contains” list of an Assembly, the contents of the Fragment are inserted into the output document at that point. A Fragment may be referred to by many FragmentParts, allowing the same information to be included in many documents without requiring users to rewrite or to copy-and-paste.

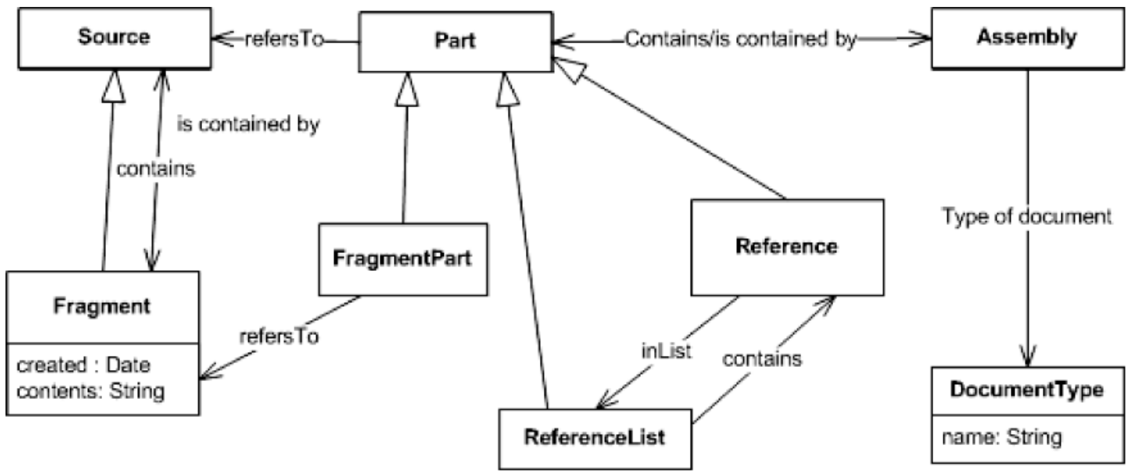

Figure 3: Classes and properties for describing the assembly of fragments and other sources into a document instance

Reference and ReferenceList are used together. An Assembly may contain a ReferenceList. Each ReferenceList contains an ordered list of Reference objects. Each Reference object has exactly one value for the "refers to" property. This value must be an instance of Source, which means that it can be a Fragment, a File, a WebSource, or an UpdateFragment. The expectation is that References will most often be associated with a File or a WebSource.

ObjectInsert is intended to allow the inclusion of values from some object outside the knowledge management ontology in a document. The specific use-case is the presentation of the definition of a requirement, along with the current status of that requirement. As noted earlier, the concept of a Requirement is part of the Organization ontology, while status reports would be one or more Fragments. In order to display an object of an arbitrary class, it is necessary for an ObjectInsert to contain two pieces of information. First, the ObjectInsert must have a property whose value is the object to display. This requirement is filled by the "refersTo" property inherited from the Part class. Second, each ObjectInsert needs a property that describes what properties of the "refersTo" object should be displayed.

This information is contained in the value of the "viewLens" property. That value is an instance of http://www.w3.org/2004/09/fresnel#Lens. The Fresnel ontology is imported from the Haystack project (http://haystack.lcs.mit.edu/). Since Fresnel is also used as part of the DocumentType section of the knowledge management ontology, it will be discussed in more detail below.

Together, the classes of Assembly, Part, ObjectInsert, Reference, and ReferenceList provide a basis for defining the contents of a specific individual document. However, it is recognized that constructing a document by repeatedly instantiating these classes is too tedious to be practical. What is needed is some way to define a general structure for a type of document and then allow the system to construct the assembly based on that structure, possibly with some assistance from a user. The concepts for the document type definition are discussed in the next section.

4.3 DocumentType

Each instance of DocumentType and its associated classes defines the structure for a category of documents. For example, a project team may define the elements of its decision document as a set of objects and properties based on classes in this section of the ontology. The DocumentType is intended to support tools that automatically create an assembly of Fragments for presentation, or that construct a user interface that enables user interaction to support the creation of documents. This part of the ontology is still under consideration; this section is based on current ideas and is subject to change.

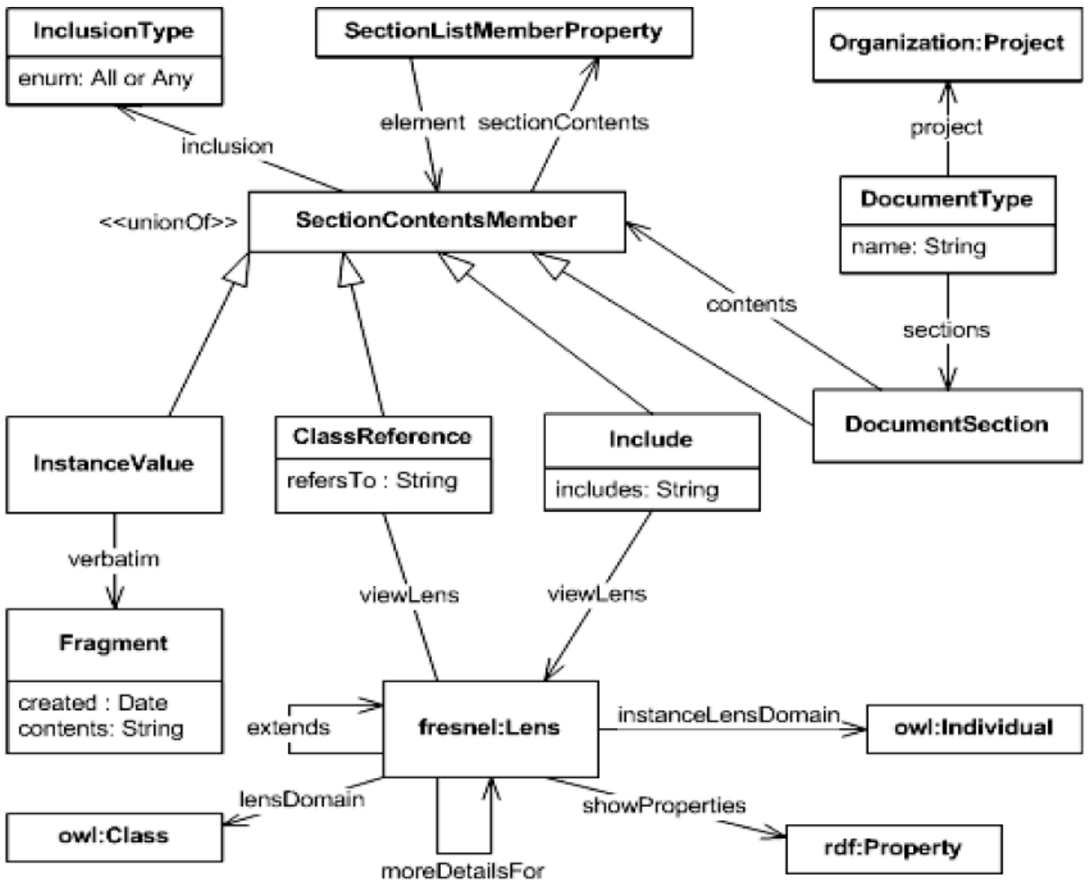

Figure 4: Classes and properties to describe types of documents

Each DocumentType has a name and zero or more associations with an org:Project (i.e., a class imported from the Organization ontology). This allows each project to potentially have its own structure for documents without requiring the invention of unique yet meaningful names for document types that have similar purposes but different requirements for multiple projects.

The structure of each DocumentType is defined by the “sections” property, which is an ordered list of DocumentSection objects. The elements of the “section” list are instances of SectionContentsMember, which is a class that is the union of several other classes: DocumentSection; Include; ClassReference; and, InstanceValue. Together, these four classes provide the means to describe each type of section that a document can have. Each of these classes is discussed in greater detail below.

Each instance of SectionContentsMember has a “sectionContents” property which is an ordered list of SectionListMember objects. This list contains objects defining the contents of the section. Each SectionListMember has an “element” property whose value is another instance of SectionContentsMember. This means that each section of a document type definition potentially consists of a list of other sections. In practice, these lists are likely to contain one member in most cases, but conceptually this structure allows a document type to have nested sections. SectionListMember has another important property: “sectionProperty,” which relates the section to a property of the object whose values are being displayed in the containing section.

As an example that will hopefully clarify the need for this property, consider the case of a list of requirements, each of which has a list of tasks, activities, and sub-requirements. A document type structure that defines how to organize a list of requirements into a document needs to be able to state that the presentation of a given requirement includes a section that displays the list of tasks. To allow this, it is necessary for the document type definition to refer to the property that relates a requirement object to the list of objects that it contains. The “sectionProperty” property fulfills this need.

It should be noted that the current definition of “sectionProperty” states that its range is rdf:Property. This means that the ontology as a whole is in OWL:Full, the most general form of OWL. OWL:Full ontologies are theoretically undecidable, and as a result many reasoners will not process them correctly, if at all. This may become an issue in the future. If so, then this decision will need to be reconsidered.

A DocumentSection has an ordered list of instances of SectionContentsMember. Each DocumentSection defines the structure of one section of the document, where the meaning of “section” is determined on a per-document type basis.

SectionContentsMember has four subclasses: InstanceValue; ClassReference; Include; and, DocumentSection. A DocumentSection can include other DocumentSections, allowing reuse of structures in multiple document type definitions.

Each instance of InstanceValue has a “verbatim” property whose value is an instance of Fragment. This Fragment will be inserted into every Assembly generated from this instance of DocumentType. This could be used for introductory material such as the description of a project, for example, or for some boilerplate text.

Include provides the ability to insert an instance of some class into a document at a given point. Include has one property, “includes”, which is the URI of the class. In an automatically generated assembly, an instance (or all instances) of this class would be inserted. For a tool that incorporates user interaction, the presence of an Include instance in the DocumentType would signal the need to display a list of available instances of that class, possibly along with a form to create a new instance.

ClassReference is very similar to Include, except that ClassReference also allows the addition of a Fragment to update the object. ClassReference would be used, for instance, to display a Task (part of the Organization ontology) along with the current status of the Task.

Both Include and ClassReference are not yet complete. In many cases, there will be a need to specify more information about the properties and values of an object that should be displayed. An appropriate approach for representing information is still under consideration.

SectionContentsMember also includes an optional “inclusion” property which, if present, will have one or two values: Any; or, All. The intent of “inclusion” is to define how many instances of a class should be inserted into the document. If the inclusion type is “All”, then every existing instance should be included. When the type is “Any”, there is a choice in which the user must participate. An example of “All” would be in the preparation of a document that lists the current status of the project: in such a document, every instance of Requirement must be included. “Any”, on the other hand, might be used in a document for a weekly meeting where the status of only some requirements has been updated.

5. Topics, TopicTargets, and External Ontologies

In addition to the construction of documents and the representation of information in a manner that allows automated repurposing, the knowledge management ontology also includes the ability to add indications of the topic of Fragments, Parts, and Assemblies. This will enable semantically-based searches in addition to traditional text-based searches, which in turn will provide the ability to generate ad-hoc documents without the need for the creation of a DocumentType structure.

Source, Part, and Assembly are all subclasses of TopicTarget. More specifically, TopicTarget is the union of the three classes, meaning that every instance of TopicTarget is also an instance of one of these classes. (Generally, OWL classes can overlap, but in this case Source, Part, and Assembly are declared to be disjoint with each other, meaning that they do not share objects.)

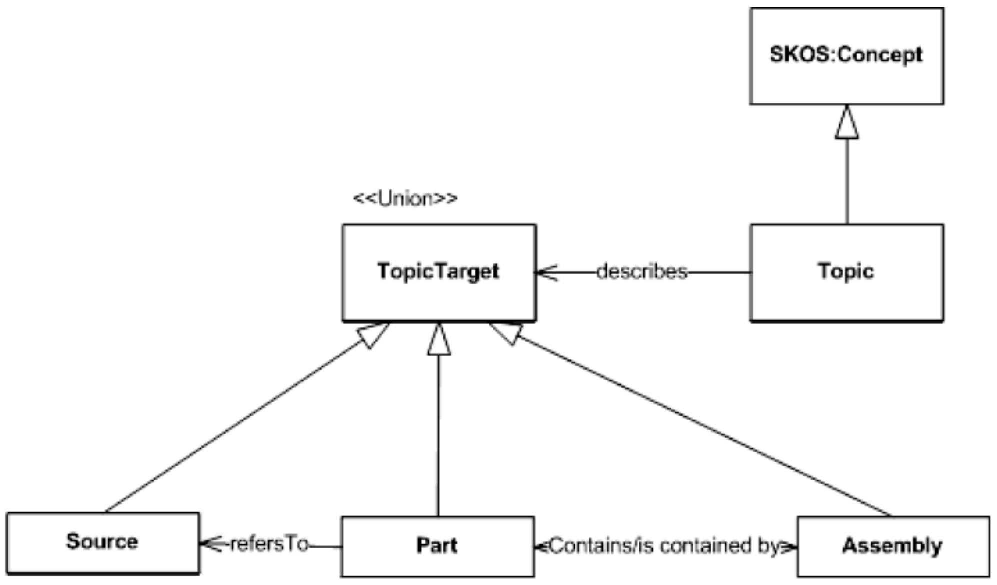

Figure 5: Classes and properties to annotate information objects with topics

A TopicTarget “is described by” one or more Topics. (“describes” is the inverse property of “is described by”.) Topic is a subclass of the SKOS class Concept. SKOS (Simple Knowledge Organization Scheme, http://www.w3.org/2004/02/skos/) is an existing ontology that is used by the knowledge management ontology (*km*). One of the useful facilities of OWL is the ease with which external ontologies may be integrated within new ontologies, potentially allowing existing tools to be used in processing parts of the new ontology. A later section of this document discusses the external ontologies utilized by the Organization and Knowledge Management ontologies.

SKOS defines a number of relationships between Concept instances. For our purposes, the most useful are likely to be SKOS:narrower and SKOS:broader, which allows the definition of subclass/superclass relationships between instances of Concept, and SKOS:relatedTo, which is a general relationship from which more specific non-hierarchical relationships may be derived.

As users enter new fragments into the repository, they will have the opportunity to add Topic instances to each fragment. These Topics may be selected from a list of existing topics or the user may enter a new topic. For new topics, the system will work with the user to determine if the topic has any relationship to existing topics. In this way, an ontology of topics will be constructed based on continuing interaction with the users of the system. Some automated processing may also be possible, based on statistical analysis of the text and the use of existing taxonomies such as WordNet (Fellbaum 1998).

Because the list of topics is created as new information is added to the system instead of relying on a pre-defined ontology, the process is related to the concept of a “folksonomy”: a word coined by information architect Tom Vander Wals (http:vanderwals.net) to refer to vocabularies that arise from the efforts of groups of people informally describing information resources. Folksonomies are generally associated with systems that allow on-line communities to add “tags” (i.e., which are essentially keyword descriptions) to various kinds of web resources. Among the better-known of such sites are del.icio.us, Technorati (www.technorati.com), and Flickr (www.flickr.com), which focus on annotating web sites, web logs, and online photographic collections, respectively.

Tag vocabularies are built one word at a time as users decide for themselves the best way to describe a resource. This practice leverages the fact that people in a specific field or area of interest tend to use similar terminologies in talking about their subject. A tagging system externalizes these vocabularies and makes them available for use in retrieving information. However, tagging systems tend to result in “flat” vocabularies where tags (or “concepts”) are not related to each other, but only to the web pages, photographs, or other resources that the tags describe. Without associations between terms, flat vocabularies are restricted to information retrieval based solely on the exact matches of key words.

The SKOS ontology retains the attractive property of a vocabulary emerging from the interaction of groups of people instead of limiting users to a predefined set of terms. At the same time, SKOS allows systems to go beyond a flat vocabulary by adding the ability for users to construct hierarchies of concepts as in a true taxonomy and to create new kinds of relationships as appropriate for specific domains. The addition of hierarchical relationships allows search and retrieval engines to understand, for instance, that a search for “mammal” should also return resources tagged with “elephant” and “mouse,” even though those resources are not tagged specifically with “mammal.”

Through the use of the Topic and SKOS ontologies, we hope to be able to provide tools to permit the retrieval of relevant information for ad hoc queries over the current state of the repository as well as for the construction of new documents.

6. External Ontologies Included in Project

One of the benefits of using semantic web languages to define ontologies is the ease with which the ontology designer can make use of existing ontologies. This benefit is increased due to the common practice of making ontologies publicly available for use in other projects.

For this project, there are three major external ontologies that supply important concepts and relationships. There are also other ontologies involved, but their influence is indirect or circumstantial.

- ***FOAF:*** Friend of A Friend (http://xmlns.com/foaf/0.1/) provides the base concepts of Person and Organization. FOAF is a widely-used ontology and it is possible that in the future some existing tools for processing FOAF information may be useful in the knowledge management system.
- ***SKOS:*** Simple Knowledge Organization System (http://www.w3.org/2004/02/skos/) provides the description of a Concept that is central to the idea of annotated documents, fragments, and other objects with topics describing their subject matter. In the *km* project ontology, skos:Concept is subclassed as km:Topic. Other aspects of SKOS are discussed in the previous section.

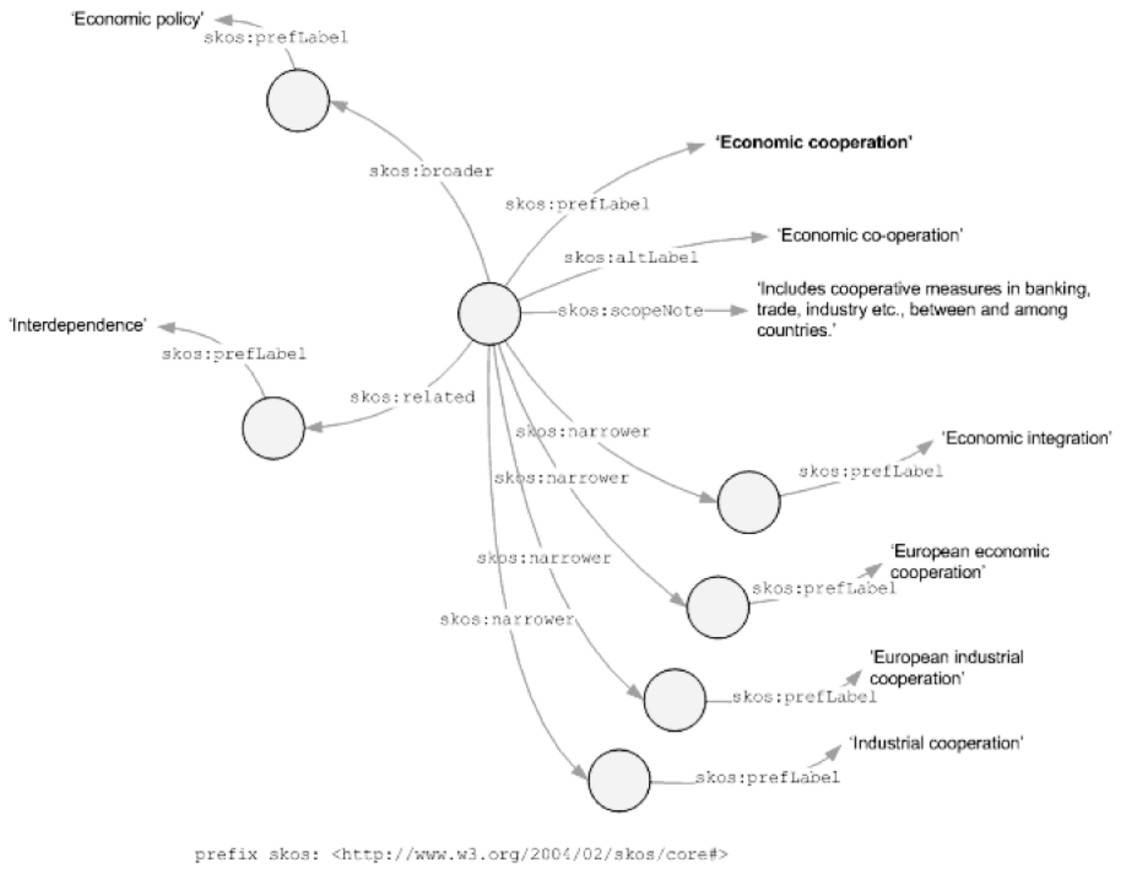

Figure 6: SKOS example
(from http://www.w3.org/TR/2005/WD-swbp-skos-core-guide-20050510/)

- ***Fresnel Lens:*** (http://www.w3.org/2004/09/fresnel#) Fresnel is a set of related ontologies whose classes and properties are intended to allow the description of documents built from values associated with objects (refer documentation at: http://simile.mit.edu/repository/fresnel/trunk/docs/manual/FresnelVocabulary.htm). There are two primary divisions in Fresnel: Lens, which defines views on objects; and, Style, which provides concepts related to the presentation of information defined using instances of Lens and related classes, and their properties.

The *km* ontology makes heavy use of Fresnel Lens in describing document type definitions. There is a possibility that the Style section of Fresnel may become the basis for future document presentation tools and user-interface elements.

There are several other ontologies that are partially included in the knowledge management system due to the fact that they are referenced by one of the three explicitly imported ontologies. Two of the more important are:

- ***Dublin Core (dc):*** (http://purl.org/dc/terms/ and http://purl.org/dc/elements/1.1/, with documentation at: http://dublincore.org/) Dublin Core defines metadata elements such as dc:title, dc:subject, dc:author, and so on, that enable descriptions of information about resources such as documents and web pages.

- ***PIM/Contact:*** (http://www.w3.org/2000/10/swap/pim/contact#) We have not been able to locate a great deal of documentation on PIM Contact. FOAF derives several of its classes, such as FOAF:Person, from classes in Contact, which links the Organization part of this project's ontology indirectly to Contact. The purpose of PIM Contact is to represent all of the objects and relationships that might be involved in defining all of the ways that a person can be contacted: email, phone (business, fax, and home), location (several types), and so on.

The connection between FOAF and PIM/Contact exposes one potential pitfall in reusing existing ontologies. In Contact, a "SocialEntity" can be related to an instance of the "_EmailAddress" class using the "emailAddress" property. In FOAF, in contrast, the "mbox" property relates an "Agent" to a "Resource." There is no explicit relationship between contact:SocialEntity and FOAF:Agent. The same information (i.e., the email address of a person or organization) is represented in two different ways and the importing ontology (FOAF) does not include an attempt to reconcile these differences. This can lead to some apparent inconsistencies in the ontology definition, as the next section discusses.

7. Issues of Working with External Ontologies

As described in the previous section, the Knowledge Management and Organization ontologies utilize several ontologies developed by other organizations. There are several positive aspects to using externally developed ontologies. For instance, the public nature of such ontologies increases the possibility that logical flaws have been removed through wider inspection, much as open source software may have fewer bugs than closed source software, simply because more people have examined the code. Also, in some cases, it may be possible for a system to incorporate existing ontology-specific tools and libraries, reducing development costs and increasing reliability. However, there are also drawbacks to using externally-developed ontologies. This section lists some issues that have arisen during development of the knowledge management system.

Overlapping Features: Some of the existing ontologies have overlapping features. In addition to the email address example mentioned above, FOAF and Contact have several other classes and properties that represent the same concepts, but have incompatible definitions. Both FOAF and Contact, for instance, define a Person class (FOAF:Person is a subclass of contact:Person) and they both define “firstName” properties for instances of their own Person class. The FOAF “firstName” property has no defined relationship to the Contact “firstName” property. This sort of condition makes it difficult to know how to handle these properties. For the purposes of the Organization ontology, the approach is to simply ignore features of external ontologies that are overlapping, conflicting, or simply outside the needs of the proposed system. There is a concern, however, that this approach may cause problems in the future in the event that there are opportunities for communication with other systems that may use these external ontologies to support their own functionality.

Different Representations: Many of the existing ontologies are expressed in RDF (and RDFS) instead of OWL. This can result in a need for extra processing, especially in the absence of an inference engine. For example: RDF includes a Property class that can be used to specify associations between two objects, or between an object and an instance of a data type, such as a string or an integer. OWL specializes rdf:Property into owl:ObjectProperty (i.e., when the value of the property is an object) and owl:DatatypeProperty (i.e., when the value is a simple data type). If an rdf:Propery has a range specified, an OWL-aware reasoner may be able to infer that the property's type is either owl:ObjectProperty or owl:DatatypeProperty, which in turn allows the property to be processed by code and rules aimed at the OWL property types. In the absence of a reasoner, this inference must be performed explicitly in order for the system to treat an instance of rdf:Property as an instance of one of the OWL property types.

Inferencing Obstacles: Ontologies, such as FOAF, need to be edited in order to be used with an inference engine. FOAF imports the OWL and RDF ontologies, and this creates errors when fed to a reasoner that already includes the semantics of these languages. Other ontologies include definitions that cause different problems for a reasoner. The Fresnel Lens ontology (http://simile.mit.edu/repository/fresnel/trunk/ontologies/lens-core.rdfs.n3) includes the description of two resources that are instances of rdf:resource, which, although legal in an RDF ontology, causes errors when loaded as an OWL ontology. (In fact, the Lens ontology defines these as instances of “rdf:Resource,” which is incorrect due to the capitalization of “Resource.” After correction, we see the error due to instantiation of RDF:resource.) Many of the ontologies included in this project have similar issues, some of which require that the ontologies be altered before they can be used.

Additionally, there is a further problem specific to the Fresnel ontology. The Fresnel ontology is in fact composed of four ontologies, namely: lens; lens extended; style core; and, style extended. Each of these ontologies is defined in its own file, but all use the same URL (i.e., http://www.w3.org/2004/09/fresnel#). This is legal for OWL and RDF, since there is nothing in the specification of either language that requires an ontology to be defined in a single file. However, from a practical point of view this arrangement is problematic because the OWL library that this project uses locates ontology representations by URL: either directly, by assuming that the URL is a location where a program can find the ontology; or, indirectly through a mapping between the URL and the local file location. In the case of Fresnel, neither of these approaches works, since there is nothing located at the given URL and it is impossible to map a single URL to four different files. Fortunately, at this early stage of our project the knowledge management ontology uses only concepts from the lens ontology part of Fresnel, which is contained in a single file. This is perhaps an illustration of how easy it can be to build dependencies into an ontology. In this case, dividing the ontology into four separate files creates no issues for the project using Fresnel (Haystack, http://haystack.csail.mit.edu/), but this decision creates problems when trying to use the ontology in other environments, creating an interoperability issue.

8. Future Work

The semantic web knowledge management ontology is sufficiently complete to serve as the basis for a set of project management and decision support tools. The first generation of such tools falls into several categories. The level of effort required to implement this list of tools is far greater than is possible to complete in a single year. Therefore, the list should be read as potential functionality for some future date, not necessarily within the next year.

1. ***Creation of instances of organization-related classes and relationships:*** These include projects, personnel, requirements, tasks, activities, and events. The implementation of the user-interface in this area has begun, and needs to be completed.

2. ***User-interface facilities:*** Development of a user-interface for constructing fragments based on initial document structure definitions. As an example, meeting minutes may have a standard structure, with variations for specific projects or even specific meeting types. Those structures will be defined as document types within the ontology. There is then a need for tools to create user interfaces for given document structures. These will probably be a combination of generic elements that apply to any document structure instance, with some specialized elements for parts of an individual document structure. The tools that are needed here fall into two areas: first, the “engine” that translates document structure definitions into user interface elements; and second, developer tools for defining special case elements as necessary.

3. ***Definition of document type structures:*** Initially, work on the two categories listed above can proceed based on hand-built models. But these structure definitions are too complex to be constructed directly. To be successful, the system must enable users who know nothing about the details of the ontology to define their own document types. This will require experimentation to determine what tools might be usable in this area.

4. ***Support for multiple drafts of a document:*** When the system has brought together many fragments into an assembly based on a document type structure definition, it is likely that the result will not be a completely polished document. Users will probably need to edit the results. But the system needs to maintain links between the various versions of each fragment. When other documents include the same fragment, every version of that fragment should be available to the user.

5. ***What-You-See-Is-What-You-Get (WYSIWYG) support:*** The final output of an instance of a document type should be acceptably formatted. The system needs to

include tools to allow the user to describe the visual presentation of the material and to associate that description with the document type structure for later use.

6. ***Searching based on semantic annotations:*** This is the original impetus for the system, although laying down the infrastructure has become the focus of work to this point.
7. ***Ad hoc document creation:*** While many work products can be described in generic format, many documents will be created only once. Creating these documents should not require the user to first create a document type structure definition, particularly since such definitions are likely to be complex. Instead, there must be some simpler interface for users to define what they want to include in a document and build the contents based on that definition.
8. ***Storable, sharable queries:*** In constructing reports, it should be possible to create queries to assemble all the relevant parts of the document contained in the repository. By making these queries persistent, another user will be able to see the same report by executing the stored queries.

References

Fellbaum C. (1998); 'WordNet: An Electronic Lexical Database'; MIT Press, Cambridge, Massachusetts.

Antoniou G. and F. van Harmelen (2004); 'A Semantic Web Primer'; MIT Press, Cambridge, Massachusetts.

OWL (2004); OWL Web Site: http://www.w3.org/2004/OWL/