Specifying Ontology Views by Traversal

Natalya F. Noy and Mark A. Musen

Stanford Medical Informatics, Stanford University,
251 Campus Drive, x-215, Stanford, CA 94305, USA
{noy, musen}@smi.stanford.edu

Abstract. One of the original motivations behind ontology research was the belief that ontologies can help with reuse in knowledge representation. However, many of the ontologies that are developed with reuse in mind, such as standard reference ontologies and controlled terminologies, are extremely large, while the users often need to reuse only a small part of these resources in their work. Specifying various views of an ontology enables users to limit the set of concepts that they see. In this paper, we develop the concept of a *Traversal View*, a view where a user specifies the central concept or concepts of interest, the relationships to traverse to find other concepts to include in the view, and the depth of the traversal. For example, given a large ontology of anatomy, a user may use a Traversal View to extract a concept of *Heart* and organs and organ parts that surround the heart or are contained in the heart. We define the notion of Traversal Views formally, discuss their properties, present a strategy for maintaining the view through ontology evolution and describe our tool for defining and extracting Traversal Views.

1 Ontology Views

Ontologies constitute an integral and important part of the Semantic Web. For the Semantic Web to succeed, developers must create and integrate numerous ontologies, from general top-level ontologies, to domain-specific and task-specific ontologies. One of the original motivations behind ontology research was the belief that ontologies can help with reuse in knowledge representation [5]. By virtue of being formal and explicit representations of a domain, ontologies could represent shared domain descriptions that different applications and agents use. When a person developing a new application chooses to reuse a published shared ontology of the domain rather than to create his own model, he gains a number of advantages: not having to reinvent the wheel, using an ontology that has already been tested in other applications, and, perhaps, most important of all, tremendously facilitating the path to information integration among different applications that use the same ontology.

Currently, there are several efforts to develop standard reusable ontologies in various domains and to make them available on the Semantic Web: from the generic upper-level ontology of SUO[1] and lexical corpus of WordNet [3] to

[1] http://suo.ieee.org/

domain-specific ontologies such as UMLS [9]. Consider for example the Foundational Model of Anatomy (FMA)—a declarative representation of anatomy developed at the University of Washington [15]. The ontology represents the result of manual and disciplined modeling of the structural organization of the human body. While the FMA ontology is a relatively recent development, many in medical informatics already consider it to be a tremendous resource that will facilitate sharing of information among applications that use anatomy knowledge. However, the size and complexity of the ontology is immense: approximately 70,000 concepts at the time of this writing. As a result, the project authors often get requests for self-contained portions of the ontology that describe only specific organs and organ parts.

In general, while providing a shared, tested, and well-accepted vocabulary, many of the large standard resources pose a formidable challenge to Semantic Web users: these resources are huge, often containing tens of thousands of concepts. However, many Semantic Web users need only a small fraction of the resource for their application. Currently such a user still needs to make sense of the complete resource, importing it as a whole, and dragging along this extra "baggage" of concepts, most of which he is never going to use. In addition to the cognitive challenge of understanding the large resource (or at least figuring out which swaths of it are not relevant and can be safely ignored), there can be a computational penalty as well if the user needs to perform reasoning with the whole ontology.

Therefore, users need the ability to extract self-contained portions of ontologies and to use these portions in their applications. For example, we can imagine the user of the FMA specifying that he needs everything that is directly related to the *heart* as well as definitions of all organs and organ parts that surround the heart. The user may also ask to include organs and organ parts that are "twice removed" from the heart. Or, the user may ask for everything related to the heart concept—a transitive closure of all relations in which the concept participates.

The notion of creating a self-contained portion of a resource has long been an area of research in databases. A database *view* provides exactly that: users specify a query that extracts a portion of database instances satisfying the query, creating a specific *view* on the data in the database. Similarly, we call a portion of an ontology an **ontology view**.

In databases, a view is specified as a query: all instances satisfying the query constitute the view. Current research on ontology views (see Section 6) takes a similar approach: an ontology view is specified as a query in some ontology-query language. However, this query-based approach does not allow users to specify a portion of an ontology that results from a particular traversal of ontology links, as in the *heart* example above. Therefore, we suggest a complementary way of defining ontology views: by **traversal specification**. In such a specification, we define a starter concept or concepts, the "central" or "focus" concepts in the result, and the details of the relationship to traverse, starting at these concepts. We call such a view a **Traversal View**.

We believe that various types of view definitions—query-based and traversal-based, and perhaps several others—are necessary to provide full flexibility to Semantic Web users in extracting specific parts of an ontology. We are developing a suite of tools that provides a convenient interface to allow users to specify views in different ways, to examine the resulting views, and to use them in their applications. In this paper, we focus on Traversal Views, defining their properties, describing an algorithm for computing them, and presenting our user-centered tools for specifying and using them.

More specifically, this paper makes the following contributions:

– defines the approach of specifying ontology views through traversal of concepts
– presents an algorithm for finding the results of traversal-view definitions
– identifies properties of Traversal Views
– presents a strategy for maintaining Traversal Views through ontology evolution
– describes a user-oriented tool for specifying Traversal Views

2 Specification of Traversal Views

In this work, we mainly consider ontologies expressed in RDF Schema and therefore consisting of classes, their properties, and instances [18]. Although, as we discuss in Section 3, our approach can be easily adapted to ontologies expressed in OWL (in fact, our implementation, which we describe in Section 5, works both with RDFS and OWL ontologies).

We combine all the properties related to a class or an instance in a **concept definition**:

Definition 1 (Concept definition). *Given an RDF class or individual C, the definition of C, $Def(C)$, is a set of RDF triples such that:*

– *any RDF triple with C as a subject is in $Def(C)$*
– *for any property P such that C is in the domain of P (as defined by $rdfs:domain$), any triple with P as a subect is in $Def(C)$*

No other triples are in $Def(C)$.

This definition implies for example, that a definition for a class C includes all the property domain statements for all the properties that contain C in their domain. Similarly, they include all the triples specifying property values for C.

We can now define the notion of Traversal Views. Traversal Views help users exploring a new ontology to find its subset that covers a particular "topic." For instance, one may not be interested in the entire human anatomy, but just in the heart and the organs that are physically close to it (say, separated from it by at most one other anatomical structure). Traversal Views enable users to extract a manageable and self-contained portion of the ontology relevant to their needs, allowing them to use and extend it for their own purposes.

Intuitively, we specify a Traversal View by specifying a starter concept (e.g., *Heart*), a list of relationships (property names) that should be traversed and the maximum distance to traverse along each of the relationships.

Example 1. We can specify a traversal view that starts at the class *Heart* and traverses the *partOf* relationship for 2 steps and the *containedIn* relationship for 1 step. We then compute the view in the following way:

1. *Heart* is included in the view
2. If *Heart* is in the domain of the property *partOf* or the property *containedIn*, then all classes in the range of these properties are also included in the view
3. If *Heart* was also an instance (RDF Schema does not prevent classes from being instances as well) and had a value for properties *partOf* and *containedIn*, those values are also included in the view

The view resulting from this process contains the traversal along the *partOf* and *containedIn* properties of length 1. We then apply the same procedure to all classes in the resulting view but consider only the *partOf* property to compute the traversal along the *partOf* property of length 2 (Recall that we requested a traversal of length 1 along the *containedIn* property.)

We now present a formal definition of a traversal view. The definition consists of two parts: the view specification (Definitions 2 and 3) and the view computation (Definitions 4 and 5).

Definition 2 (Traversal Directive). *A* ***traversal directive*** *D for ontology O is a pair $\langle C_{st}, \mathcal{PT} \rangle$ where*

- *C_{st} is a class or an instance in O (the starter concept of the traversal);*
- *$\mathcal{PT}$ is a set of* ***property directives****. Each property directive is a pair $\langle P, n \rangle$, where P is a property in O and n is a nonnegative integer or infinity (∞), which specifies the depth of the traversal along the property P. If $n = \infty$, then the traversal includes a transitive closure for P starting with C_{st}.*

In Example 1 we defined a single traversal directive, with *Heart* playing the role of the starter concept C_{st} and two property directives in $\mathcal{PT}$: $\langle partOf, 2 \rangle$ and $\langle containedIn, 1 \rangle$.

Definition 3 (Traversal View Specification). *A* ***Traversal View specification*** *T is a set of traversal directives $\mathcal{TD}$*

The view specification in Example 1 contained only one traversal directive.

Example 2. Suppose in addition to the concept *Heart* and its related concepts from the view in Example 1, we also want to include all parts of the lung in the view (without limiting the depth of the traversal). We will then include the following directive into the view specification (in addition to the directives in Example 1):

- $C_{st} = Lung$

– $\mathcal{PT} = \{\langle has\ part, \infty\rangle\}$

We define the procedure for computing Traversal Views recursively as follows.

Definition 4 (Traversal Directive Result).
*Given a traversal directive $D = \langle C_{st}, \mathcal{PT}\rangle$ for an ontology O, a **traversal directive result** $D(O)$ (the result of applying directive D to O) is a set of instance and class definitions (in the sense of Definition 1) from O such that*

1. *C_{st} is in $D(O)$ (the starter concept is in the result)*
2. *For each property directive $D_s \in \mathcal{PT}$, where $D_s = \langle P, n\rangle$, $n > 0$,*
   - *if C_{st} is a class in the domain of the property P, and a class $C \in O$ is in the range definition for P at C_{st}, then C in $D(O)$*
   - *if C_{st} is an instance and it has a value for the property P, and this value is another class or instance $F \in O$, then F is in $D(O)$*
3. *$\mathcal{PT}_{next}$ is a set of property directives such that for each property directive $D_s = \langle P, n\rangle$ in $\mathcal{PT}$, the directive $\langle P, n-1\rangle$ is in $\mathcal{PT}_{next}$. If $n = \infty$, then $n - 1 = \infty$. For each class or instance F that was added to $D(O)$ in step 2, the traversal directive result for a traversal directive $D_F = \langle F, \mathcal{PT}_{next}\rangle$ is in D(O).*

No other concepts are in D(O).

This definition describes the procedure that we followed in finding the view in Example 1. Note that along with the concept itself, we include its definition in the view, which includes the property values for the concept, as well as range, cardinality, and other constraint statements for the properties for which the concept serves as a domain.

Finally, we define the concept of a traversal view.

Definition 5 (Traversal View). *Given an ontology O and a Traversal View specification T, consisting of a set of traversal directives $\mathcal{TD}$, a **traversal view** $TV(O,T)$ is the union of traversal directive results for each traversal directive $D \in \mathcal{TD}$*

In other words, a Traversal View is a subset of an ontology that consists of classes and instances on the path of the traversal specified in the view.

Note that if the user wants to specify a view that contains the starter concept and everything that is related to it—the transitive closure of all relations emanating from the starter concept—he needs to specify all the properties in the ontology and set the infinite depth of traversal on all of them (The user interface that we describe later makes this operation easy by having a single **"everything"** button.)

Standard properties. In addition to the properties defined in the ontology, traversal directives can include standard relations such as *rdf:subclassOf*, *rdf:type*, and so on. In fact, a useful directive could include a starter concept, and some specific number of levels of its subclasses (or superclasses). To include these standard ontology relations in the traversal directives, internally, we treat them as properties: Each class has a property *direct-subclasses*; the values of this property are all subclasses of the class. Similarly, there are properties *direct-superclasses*, *direct-instances*, *type-of*.[2] Therefore, these properties can be used in traversal directives just as all the regular properties.

View Boundary. When we extract a portion of an ontology for the view, we inevitably leave some of the definitions in the ontology incomplete. For instance, a view may include a class C_1 but not a class C_2 which is a range for a property P for C_1. Therefore, the definition of C_1 in the view is incomplete. At the same time, we may not be able to include C_2 in the view since it will violate the specific directives in the view definition.

In this case, we say that the concept C_2 in this example is in the **boundary** of the view definition: it is referenced by one of the concepts in the view but it is not itself included in the view. Maintaining the list of concepts in the view boundary is extremely useful for interactive specification of a view: the user can see which concepts are missing from the definition of the view and add them to the specification.

Traversal Views for Description Logics formalisms. While we used RDF Schema to define Traversal Views, we can also specify views for description-logic ontologies, such as ontologies in OWL [2]. Classes and properties in OWL are similar to classes and properties in RDF Schema. Definitions 2, 3, 5 are trivially adapted by adjusting terminology.

In the Step 2 of Definition 4 we consider all properties that have the starter concept as part of their *domain*. There are a number of ways to restrict a property value in OWL, by using *allValuesFrom*, *someValuesFrom*, or *hasValue* restrictions. These restrictions have classes or individuals as their values. More specifically, we add the following concepts to $D(O)$ in Step 2 of the Definition:

– *If an object property P (a property that specifies relations between instances of classes) has a restriction for C_{st}, the classes specified in the restriction (whether for* allValuesFrom*,* someValuesFrom*,* hasValue*) are in $D(O)$*
– *If an object property P has C_{st} in its domain, the classes in the range of P are in $D(O)$*
– *If an object property P has a value for C_{st} as an individual, that value is in $D(O)$*

The view designed in such a way has all the same properties that we discuss for general Traversal Views in the next section.

3 Properties of Traversal Views

We now discuss the properties of Traversal Views that follow from Definitions 2 through 5. We consider compositional properties of single traversal directives

[2] The Protégé ontology-editing environment, which was the platform for our implementation discussed later, takes exactly this approach.

and complete views. We then discuss completeness of our definition and computational cost of computing the view. Finally we discuss the issue of using Traversal Views to answer queries.

3.1 Composition of Directives and Views

Composition of traversal directives. It is easy to show, based on Definitions 2 and 4, that traversal directives are *commutative* with respect to composition: the set of concepts in the traversal view does not depend on the order of traversal directives in the view. Each directive is applied to the complete source ontology, thus traversal directive results are not affected by other directives. The result of the view is the union of the results of applying each directive.

Traversal directives are also *distributive.* Recall that a traversal directive D consists of a starter concept C_{st} and a set of property directives. Consider two traversal directives D_1 and D_2 with the same starter concept C_{st}. It follows from Definition 4 that composition of traversal directives is distributive:

$$\langle C_{st}, \mathcal{PT}_1\rangle \oplus \langle C_{st}, \mathcal{PT}_2\rangle = \langle C_{st}, \mathcal{PT}_1 \circ \mathcal{PT}_2\rangle \qquad (1)$$

Here the composition of the sets of property directives $\mathcal{PT}_1$ and $\mathcal{PT}_2$ is the union of the directives in each set.

Composition and chaining of Traversal Views. Given an ontology O and two Traversal View specifications T_1 and T_2, we define **composition** of T_1 and T_2 as applying both T_1 and T_2 to O and then taking the union of the result:

$$T_1 \oplus T_2 = TV(O, T_1) \cup TV(O, T_2) \qquad (2)$$

Composition defined in this way is *commutative*: the results of applying directives in one of the traversal views does not depend on the other view, since both are applied to the complete ontology O.

Chaining of Traversal Views T_1 and T_2 is the result of applying T_2 to *the result of* applying T_1 to O:

$$T_1 \circ T_2 = TV(TV(O, T_2), T_1) \qquad (3)$$

Chaining of Traversal Views is *not commutative*: When the view T_2 applied to T_1, the set of concepts in T_2 is limited by the concepts that were included in T_1. These concepts may not include some of the concepts that would have been in T_2 if it was applied to all of the ontology O. In fact, the specification of the view T_2 may not even be a valid view specification since some of the starter concepts in traversal directives in T_2 may not even be in T_1. Thus, Traversal Views are not commutative with respect to chaining.

3.2 Completeness and Complexity

Completeness. Traversal Views are complete in the sense that for any subset of concepts in an ontology there is a (not necessarily unique) Traversal View that defines it. Let $\{c_1, ..., c_n\}$ be a subset of concepts. A Traversal View consisting of a set traversal directives $\{td_1, ...td_n\}$ such that $td_k = (c_k, \emptyset)$ defines exactly of the set $\{c_1, ..., c_n\}$. In other words, for each concept in the subset, we create a traversal directive containing that concept as a starter concept and no property directives.

Computational Properties. Let t be the number of traversal directives in a view definition and n be the number of concepts in the ontology. Then the running time of computing the Traversal View is $O(t * n)$. In other words, if the number of traversal directives in a view is limited by some constant c, computation of a Traversal View is linear in the size of the ontology. Indeed, computation of each traversal directive, which simply follows Definition 4, needs to examine each concept in the ontology exactly once. Suppose a concept C was examined during iteration i. If C is visited at any subsequent iterations in computing the same traversal directive, we do not need to compute the traversal directive $\langle C, \mathcal{PT}_{next} \rangle$: the depths for all the property directives will be necessarily less than at iteration i and therefore computing this directive will not add any new concepts to the view.

3.3 Using Traversal Views in Inference

Consider an ontology O and a Traversal View specification T. If the ontology O is consistent (for example, based on the interpretation of RDF Schema Semantics [4]), then the Traversal View $TV(O, T)$ also represents a consistent ontology based on this semantic interpretation: According to Definition 4, we do not add any new triples to $TV(O, T)$ that did not already exist in O. Further, given the fact that the constructs available in RDF Schema are monotonic, if a set of triples is consistent according to the RDF Semantics interpretation, then its subset is also consistent. Therefore, any relation between concepts in the view $TV(O, T)$ that is entailed by the view defintion, is also entailed by the definitions in the full ontology O.

Note that this property of views is not true in general for Traversal Views for OWL ontologies. Consider for example the notion of disjointness in OWL. If the disjointness axiom is not included in the view, we may conclude that two entities are equivalent that should not be equivalent if disjointness is considered. We are currently working on identifying a set of syntactic constraints for an OWL ontology that would allow us to assert that any relation between two entities that is entailed by $TV(O, T)$ is also entailed by O.

4 Traversal Views and Ontology Evolution

It is inevitable that ontologies change and users need to work with different versions of the same ontology. After using Traversal Views to extract a portion of an ontology, and then getting a new version of this ontology, the user should be able to determine (1) whether the view definition is still valid for the new version (that is, all the starter concepts and all the properties explicitly mentioned in the view definition are present in the new version of the ontology); and (2) whether the subset of the ontology specified by the view is unchanged.

Noy and Musen [11] have developed the PromptDiff algorithm to compare different versions of the same ontology. We integrated PromptDiff with Traversal Views to answer the questions above.

Given two versions of an ontology V_{old} and V_{new}, for each concept in V_{old}, PromptDiff determines a corresponding concept in V_{new} if there is one. (It uses a set of heuristics to find correspondences for classes and properties that changed their names or definitions). Using this information we determine whether a view T, which was defined for V_{old}, is still valid for V_{new}. If for each concept from V_{old} that is explicitly used in T as a starter concept or a property name, there is a corresponding concept in V_{new}, then the view is valid. Furthermore, if any of the concepts explicitly used in T have changed, the user has the option of updating the view T to refer to the corresponding new concepts.

If the view has been materialized, we can use a similar technique to determine if the materialized view needs to be updated by examining the view to determine whether each concept in the view has a corresponding concept in the new version.

5 Implementation

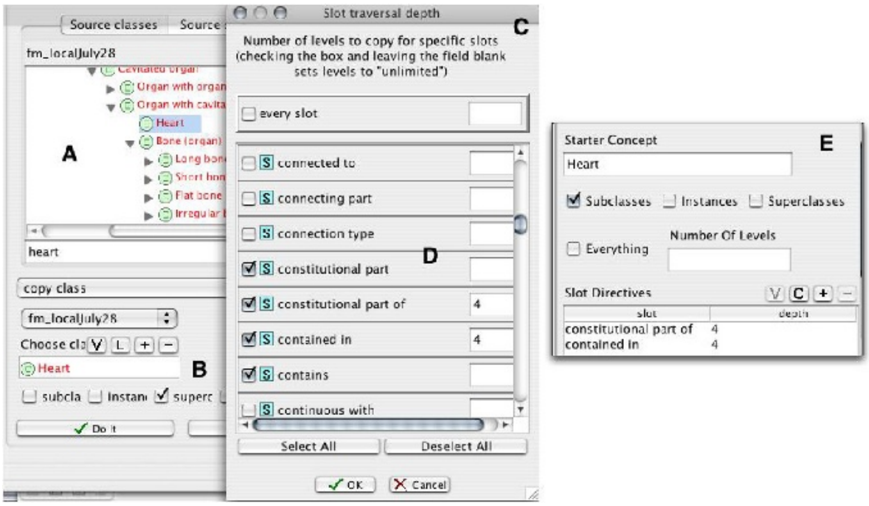
We have implemented Traversal Views as a plugin to the Protégé ontology-development environment.[3] Protégé provides an intuitive graphical user interface for ontology development, a rich knowledge model, and an extensible architecture that provides API access both to the Protégé knowledge bases and to its user-interface components.

Figure 1 presents the user interface for Traversal View specification. First, the user selects a starter concept by browsing the ontology (Figure 1A). Then the user specifies property directives. Specification may simply include checking some of the built-in relationships, such as subclasses or superclasses (Figure 1B). If the user wants to specify a more detailed directive, he does this through an additional dialog (Figure 1C), specifying the depth of traversal for each property. In addition, the user can select the "everything" option and set the same depth of traversal for all the properties in the ontology.

After the user specifies and issues a traversal directive, we materialize the directive by copying the concepts in the view to the user's local space. In addition, we save the traversal directives as instances in an *ontology of traversal directives*.

When the user saves the results of his work, both the materialized view (the extracted portion of the source ontology) and the ontology representing the traversal directive (the view definition) are saved. The user then has the option of automatically "replaying" these directives on the source ontology. Another possibility would be to "mark" concepts from the ontology that belong to the view rather than to materialize the view.

[3] http://protege.stanford.edu



fm_localJuly28
Cavitated organ
Organ with organ
Organ with cavita
Heart
Bone (organ)
Long bon
Short bon
Flat bone
Irregular b
heart
copy class
fm_localJuly28
Heart
☐ subcla ☐ instan ☑ superc

Number of levels to copy for specific slots (checking the box and leaving the field blank sets levels to "unlimited")

☐ every slot	
☐ S connected to	
☐ S connecting part	
☐ S connection type	
☑ S constitutional part	
☑ S constitutional part of	4
☑ S contained in	4
☑ S contains	
☐ S continuous with	

Starter Concept
Heart
☑ Subclasses ☐ Instances ☐ Superclasses
☐ Everything
Number Of Levels

slot	depth
constitutional part of	4
contained in	4

Fig. 1. Specifying and examining Traversal Views. The user chooses the starter concept by browsing the hierarchy (A), specifies the traversal along the built-in properties (B), brings up a separate window (C), which lists all the properties in the ontology and built-in properties (D). The user can examine and edit the defined views through a simple form-based interface of Protégé (E).

Using an ontology to store the directive is likely to lower the level of expertise required from the user to update the directive: In Protégé users browse and edit ontology instances (which is exactly what traversal directives are) through a simple form interface. To many, editing traversal directives through this interface (Figure 1E) is easier than writing an SQL query. Furthermore, a user can open the ontology with saved traversal directives in an ontology editor, such as Protégé, and use this simple interface to change the directives before applying them again.

6 Related Work

Our work is complementary to the work of using queries to define ontology views. Volz and colleagues [16,17], for instance, define a view language based on the RQL query language [7]. In this framework, a view represents a new class or property in the ontology. The authors restrict the RQL queries that can be used in the view to ensure that the view returns only unary or binary relations (classes or properties respectively).

In theory, query-based approaches to defining ontology views are equivalent to our traversal-based approach in terms of their completeness: Just as we showed that any subset of an ontology can be produced by a combination of trivial traversal directives, we can show that any subset of an ontology can be produced by a combination of trivial queries. However, depending on what the user needs in the view and what information he can specify about the view result, a query-based or a traversal-based approach may be more appropriate.

Magkanaraki and colleagues [10] take the approach of defining query-based views further. In their RVL language, which also uses RQL for querying ontologies, the authors propose mechanisms for restructuring the original class and property hierarchies, allowing the creation of new resources, property values, classes, or properties. Therefore, a view definition includes not only the query itself, but also a set of statements that define these new structures, linking them to the query results. We can take a similar approach with Traversal Views, allowing the users to rename or reorganize the concepts in the view.

XML query languages, such as XQuery [1] are also based on traversing structures. There are, however, important differences between languages such as XQuery and Traversal Views: In XQuery, the traversal is aimed at collecting the *data* themselves rather than the schema and at presenting the data in the result according to a particular structure specified in the query. Traversal Views are aimed at traversing and collecting the schema elements themselves in the result.

Researchers in the Process Interchange Format (PIF) Project have introduced the notion of Partially Shared Views as a way of allowing different groups to extend a shared standard [8]. A Partially Specified View includes of the type hierarchy and templates, which are similar to concept definition. In this sense, Partially Specified Views are reminiscent of Traversal Views: they include definitions of concepts themselves rather that their instantiations.

A combination of views as a mechanism for extracting ontology subsets and mechanisms that allow for renaming or slight reorganization of concepts in the view is crucial to encouraging ontology reuse. This reuse, in turn, can facilitate the problem of integrating ontologies. Orbst [13] suggests creating application views of ontologies as a means of enabling applications to use standard well-developed ontologies. This approach combines views or “perspectives” with mappings between concepts in the view and in the application. The task of creating the mappings becomes much easier if the ontology to map to is smaller and contains only the concepts that are relevant to the application. Ontology views as means for extracting self-contained subsets of standard ontologies can help in this task.

7 Summary and Open Issues

We have presented *Traversal Views* as a way of defining an ontology view. In a Traversal View, a user specifies a subset of an ontology to include in the view by specifying of starter concepts to include, the links to follow from those concepts, and how deep. This mechanism enables users to extract self-contained portions of an ontology related to a particular concept or a set of concepts (such as all parts and components of the Heart).

Database researchers have done a lot of work on using views directly to *answer queries* [6]. Since database views are themselves queries, this area of research centers on reformulating the user's query to express it in terms of existing views. While Traversal Views do not map to queries directly, whether we can use their definitions directly in answering (perhaps a restricted set of) queries is an open research issue.

Interfaces that enable users to browse the views while "popping out" to the original ontology when hitting the boundary of the view are also necessary. Developers may omit or forget something when defining a view and enabling users to see more of the original context for concepts in the view may be helpful.

Traversal Views enable users to extract classes and instances to include in the view, thus simplifying the original ontology for them. However, these classes and instances can have definitions that are themselves very complex. The next step would be to allow *pruning the definitions*, hiding some of the parts or presenting some the parts differently, based on the user's perspective.

We limited a *concept definition* for a concept C to RDF triples where C is the subject and to ranges of properties where C is a domain. This definition can be generalized to include a wider "neighborhood" of the concept C in an RDF graph. For example, it can include an *anonymous closure* of an RDF graph [14], which is computed by following graph edges, from subject to object of statements, until an anonymous RDF resource or RDF Literal is found. This extension will include, for example, all members of an RDF collection of values for a property of C in the concept definition of C. This, and similar, extensions to a concept definition, effect what a Traversal View ultimately includes.

A similar issue is the treatment of *transitive properties* for languages such as OWL. Suppose a Traversal View includes a traversal directive with a starter concept C_{st} and a property directive $\langle P, 1\rangle$, where P is a transitive property. If we issue a query for all concepts X such that $P(C_{st}, X)$, looking for all the concepts directly related to C_{st} through property P, we will get all concepts in the transitive closure of C_{st} with respect to P, rendering the depth of traversal meaningless. One possibility to handle limited traversal of transitive properties is to distinguish between explicit and inferred triples in the ontology definition.

As part of future work, we plan to perform user studies to evaluate different approaches to these issues to determine which ones provide Traversal Views that correspond most closely to users' intuition.

Finally, we would like to link query-based view, Traversal Views and other types of view-definition techniques in a unified framework to enable users to combine the different means of defining ontology views. Furthermore, we consider definition and manipulation of ontology views to be part of the general framework for ontology management. We have developed a suite of tools—PROMPT [12]—that supports various tasks in ontology management, such as comparing ontologies, merging them together, maintaining and comparing different versions, and so on. For example, the users can compare or merge views of different ontologies, find diffs between different versions of a view, integrate two views into a single ontology.

Acknowledgments. This work was supported by a grant from the DARPA, executed by the U.S. Army Medical Research and Materiel Command/TATRC

Cooperative Agreement, Contract # W81XWH-04-2-0012. This work was conducted using the Protégé resource, which is supported by grant LM007885 from the United States National Library of Medicine. The Prompt suite is available as a plugin to the Protégé ontology-development environment at `http://protege.stanford.edu/plugins/prompt/prompt.html`

References

1. S. Boag, D. Chamberlin, M. F. Fern‡ndez, D. Florescu, J. Robie, and J. SimŽon. XQuery 1.0: An XML query language. Technical report, W3C, 2003.
2. M. Dean, D. Connolly, F. v. Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. Web ontology language (OWL) reference version 1.0, http://www.w3.org/tr/owl-guide/, 2002.
3. C. Fellbaum, editor. *WordNet: An Electronic Lexical Database.* MIT Press, Cambridge, 1998.
4. R. Fikes and D. L. McGuinness. An axiomatic semantics for RDF, RDF-S, and DAML+OIL. Technical report, World Wide Web Committee (W3C) Note, 2001.
5. T. R. Gruber. A translation approach to portable ontology specification. *Knowledge Acquisition*, 5:199–220, 1993.
6. A. Halevy. Answering queries using views: a survey. *VLDB Journal*, 2001.
7. G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. RQL: A declarative query language for RDF. In *Eleventh International World Wide Web Conference*, page 592–603, Honolulu, Hawaii, USA, 2002.
8. J. Lee and T. W. Malone. Partially shared views: a scheme for communicating among groups that use different type hierarchies. *ACM Transactions on Information Systems (TOIS)*, 8(1):1–26, 1990.
9. D. Lindberg, B. Humphreys, and A. McCray. The Unified Medical Language System. *Methods of Information in Medicine*, 32(4):281, 1993.
10. A. Magkanaraki, V. Tannen, V. Christophides, and D. Plexousakis. Viewing the semantic web through RVL lenses. In *Second International Semantic Web Conference*, volume 2870, pages 96–112, Sanibel Island, FL, 2003. Springer-Verlag.
11. N. F. Noy and M. A. Musen. PromptDiff: A fixed-point algorithm for comparing ontology versions. In *18th Conf. on Artificial Intelligence (AAAI-2002)*, Edmonton.
12. N. F. Noy and M. A. Musen. The PROMPT suite: Interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies*, 59(6):983–1024, 2003.
13. L. Obrst, H. Liu, and R. Wray. Ontologies for corporate web applications. *AI Magazine*, 24(3):49–62, 2003.
14. M. Palmer, A. Naeve, and F. Paulsson. The SCAM framework: helping semantic web applications to store and access metadata. In *1st European Semantic Web Symposium (ESWS 2004)*, Heraklion, Greece, 2004.
15. C. Rosse and J. L. V. Mejino. A reference ontology for bioinformatics: The foundational model of anatomy. *Journal of Biomedical Informatics.*, 2004.
16. R. Volz, D. Oberle, and R. Studer. On views in the semantic web. In *2nd Int. Workshop on Databases, Documents and Information Fusion (DBFUSION02)*, 2002.
17. R. Volz, D. Oberle, and R. Studer. Implementing views for light-weight web ontologies. In *IEEE Database Engineering and Application Symposium (IDEAS)*, Hong Kong, China, 2003.
18. W3C. Resource description framework (RDF), 2000.