An Algebra for Semantic Interoperability of Information Sources

Prasenjit Mitra, and Gio Wiederhold
Infolab, Stanford University
Stanford, CA, USA 94305
{mitra, gio}@db.stanford.edu *

Abstract

Resolving heterogeneity among the various biological information systems is a crucial problem if we wish to gain value from the many distributed resources available to us. For example, information from multiple protein databases (e.g., Swiss-Prot and PDB) might need to be composed to answer queries posed by end-users. Problems of heterogeneity in hardware, operating systems, interfaces and data structures have been widely addressed, but issues of diverse semantics have been handled mainly in an ad-hoc fashion. This paper highlights the ONION (ONtology compositION) system that enables semantic interoperation among various information sources by articulating the ontologies associated with them. An articulation focuses on the semantically relevant intersection of information resources. Although the generation of articulations (semantic correspondences between the ontologies) cannot be fully automated, we take a semi-automatic approach. ONION uses heuristic algorithms for the automatic generation of suggested articulations. This paper outlines an algebra for ontology composition based on their articulations. We show the properties of the algebraic operators and how they depend upon the articulation functions that generate the articulations. Query optimization is enabled based on the properties of the algebraic operators.

1. Introduction

A large number of diverse bio-information sources are available today. The future of the biological sciences promises the generation of more and more data. No individual source will provide us with answers to the queries that we will need to ask. Instead, knowledge has to be composed from multiple sources in order to answer most queries. Even though multiple databases may cover the same data, their focus might be different. For example, even though SWISS-PROT, PDB are both protein databases, we might want to get informatioin about the sequence as well as the structure of a particular protein. In order to answer the query we will need to get data about the protein from both the sources and combine them in a consistent fashion.

On a larger scale, semantic interoperation needs to be enabled among communities, e.g. neuroinformatics and genomics data needs to be linked [1]. The Human Brain Project Home page [2] at the National Institute of Mental Health starts with the statement:

> Understanding brain function requires the integration of information from the level of the gene to the level of behavior. At each of these many and diverse levels there has been an explosion of information, with a concomitant specialization of scientists. The price of this progress and specialization is that it is becoming virtually impossible for any individual researcher to maintain an integrated view of the brain and to relate his or her narrow findings to this whole cloth. Although the amount of information to be integrated far exceeds human limitations, solutions to this problem are available from the advanced technologies of computer and information sciences.

Enabling interoperation among bio-information sources is, thus, a critical problem for sustainable and fast progress in the biological sciences.

Researchers in bioinformatics have long identified the need of interoperation among large databases, knowledge-bases and other available information sources, especially, on the World-Wide Web. Despite advances, even today, interoperation among information sources on the web in the biological sciences is enabled by hypertext links between related sources. The onus is on the end-user to navigate the links meaningfully to collect the necessary information. With the proliferation of high-quality information sources, it will soon not be possible for sources to link meaningfully to all other related sources - especially since the data and the structure of web-sources change rapidly. Not only is the process of collating information manually extremely tedious and time-consuming, but also, often, the end-user does not have any idea of the semantics used by the builder of the information source.

Therefore, we need efficient tools that enable interoperation among biological information sources with minimal amount of human intervention. Karp [3] has identified the several approaches that have been proposed and implemented by bioinformatics researchers and proposes a strategy for database interoperation. We extend Karp's approach to apply to not only databases, but also to knowledge bases and other information sources.

In this paper, we present a brief overview of the ONION (ONtology compositION) system, which takes a principled approach to enable semi-automatic interoperation among heterogeneous information sources. As in [4], [5], and [3], we assume that information sources are independently created and maintained. In Karp's system, each database comes with a schema which is saved in a Knowledge Base of Databases(KoD). Correspondingly, we assume that associated with each information source is an ontology. However, we do not require all ontologies to be saved in a central repository like the KoD.

In the recent years, several ontologies have been developed in the biological sciences e.g., [6] (for a comprehensive list see [7]). The ontologies associated with information sources are based on some existing, known vocabularies and conceptual models. Native drivers and wrappers provide access to the ontologies and help us restructure the information if needed.

We establish application-specific *articulation rules* - rules that establish correspondence between concepts in different ontologies - semi-automatically. ONION suggests articulation rules that are generated by an automatic *articulation generator*. The articulation rules are then verified by an expert and stored for use while answering queries or composing information from the sources. The system logs the expert's response, and adapts the algorithms based upon the feedback so as to produce better suggestions for articulations for similar applications.

The contributions of this work are as follows:

- ONION saves only the articulations, which are much smaller than the source ontologies or schemas (since they are application-specific), in a central repository.
- ONION uses several heuristic algorithms that suggest articulations (between ontologies), which are then ratified by an expert using a GUI tool.
- Query optimization is enabled based on an ontology-articulation algebra. We outline the properties of the algebraic operations that determine whether operands can be rearranged in order to boost performance.

The ONION approach to interoperation has several advantages. First, only as much information as is absolutely necessary is maintained in the central repository. This distributed philosophy results in a more scalable, available, and updatable system. Second, the automated articulation generator takes away a lot of manual effort for the generation of simple rules. The domain expert can then provide the more complex rules and use a simple GUI to ratify the generated ones. The system learns from the expert and generates better and better articulations. Third, the establishment of an algebraic framework enables us to get a better insight on 1) how information can be composed systematically, 2) how the articulation generator determines this composition, and 3) how queries can be optimized based on the properties of the articulation generator. To the best of our knowledge, such an algebraic framework has not been considered in prior works on interoperation.

The rest of the paper is organized as follows. In Section 2, we describe the common conceptual model that ONION uses for its internal representation of ontologies. In Section 3 we discuss the semi-automatic articulation of ontologies. In Section 4 we outline an Ontology Algebra that we use to compose information from diverse sources. Section 5 concludes the paper.

2. The ONION Conceptual Model

We need to resolve the heterogeneity among information sources to enable meaningful information exchange or interoperation among them. The two major sources of heterogeneity among the sources are as follows: First, different sources use different conceptual models and modeling languages to represent their data and meta-data. Second, sources using the same conceptual model differ in their semantics. The ONION system uses a common ontology format, which we have described below. It first converts all external ontologies to this common format and then resolves the semantic heterogeneity among the objects in the ontologies that we seek to articulate.

Melnik, et al., [8] have shown how to convert ontologies and different classes of conceptual models into those using one common format. For example, say one information source uses UML [9] and another uses DAML+OIL [10]. ONION will convert the ontologies associated with both information sources to the ONION *conceptual model* described below. Since the number of classes of such conceptual models that are in use and that we want to support is small, we will provide wrappers, which will convert from these models to the ONION format.

Information sources were, are, and will be modeled using different conceptual models. We do not foresee the creation of a *de facto* standard conceptual model that will be used by all information sources. On the other hand, we need a common ontology format for our internal representation. We use the ONION format to represent the source ontologies and manipulate them to create the articulation ontology. The design choices for the conceptual model that we will transform the various source ontologies to range from the least common denominator of the different conceptual models used by the various sources to the greatest common multiple of them. Instead of choosing a model that has various complex features that capture the intricacies of all the conceptual models, we strive to keep our model simple.

2.1. A Graph-Oriented Conceptual Model

Our common conceptual model for the internal representation of ontologies is based on the work done by Gyssens, et al.,[11]. In its core, we represent an ontology as a graph. Formally, an ontology $O = (G, R)$ is represented as a directed labeled graph G and a set of rules R. The graph $G = (V, E)$ comprises a finite set of nodes V and a finite set of edges E. The label of a node is given by a a non-null string. In the context of ontologies, the label is often a noun-phrase that represents a concept. The label of an edge is the name of a semantic relationship among the concepts and can be null if the relationship is not known. A more detailed description of the conceptual model can be found in [12].

The graph in the ONION conceptual model can be expressed using RDF [13]. Each edge in our graph is coded as an RDF sentence, with the two nodes being the subject and the predicate and the relationship being the property. However, in order to keep our model simple, we have not included the containers that provide collection semantics in RDF. If the children of a node need to be ordered we use a special relationship, as explained below. By choosing RDF, we can use the various tools that are available and do not have to write parsers and other tools for our model.

The set of logical rules R, associated with an ontology, are rules expressed in a logic-based language. Although, theoretically, it might make sense to use first-order logic as the rule language due to its greater expressive power, to limit the computational complexity we will use a simpler language like Horn Clauses.

2.2. Semantic Relationships and Articulation in ONION

Certain conceptual models allow only strictly-typed relationships with pre-defined semantics. For instance, relationships like SubClassOf, AttributeOf, etc., have very clearly defined semantics in most object-relational databases. Other models allow any user-defined relationships without any restriction. For instance, relationships like $OwnerOf$ tend to be interpreted according to the semantics associated to it by the local application.

If the ONION articulation generator understands the semantics of the relationships used in the two ontologies it is articulating, the articulation generated by it is more precise. User-defined relationships are not be type-checked or interpreted by the system, since it does not know of their application-specific semantics. However, the user while defining relationships, may provide a set of rules that specify the semantics of those relationships. For example, if the source ontology uses a relationship IsA and has a rule that says that IsA" is transitive, the articulation generator can, then, use the information to generate better matches.

The articulation generator generates matches among nodes in the two source ontologies that is supplied to it and does not attempt to match relationships among ontologies. The articulation rules that the articulation generator generates uses only the relationships whose semantics are predefined to establish correspondences among nodes in the source ontologies. The set of relationships with pre-defined semantics is $\{SubClassOf, PartOf, AttributeOf, InstanceOf, ValueOf\}$. The ONION conceptual modeling encourages the use of a set of strictly-typed relationships with precisely defined semantics, while allowing user defined relationships whose semantics are not interpreted by the system.

In ONION, we assign the conventional semantics to each of these relationships. Some of these relationships impose type-restrictions on the two nodes they relate. Some of the relationships (like $SubClassOf$, $InstanceOf$) are somewhat similar to those in RDF-Schema but the set of relationships that have defined semantics in our conceptual model is small so as to maintain the simplicity of the system.

The following is a description of the semantics of he set of pre-defined relationships available in our common conceptual model:

SubClassOf: The relationship is used to indicate that one concept is a subclass of another. The two concepts that it relates must be of type Class. For example, $(Aminopeptidase\ SubClassOf\ Enzyme)$. That is any instance of the class $Aminopeptidase$ is also an instance of the class $Enzyme$. All the attributes of the class $Enzyme$ are also attributes of the class $Aminopeptidase$. The system interprets the relationship $SubClassOf$ as transitive.

AttributeOf: This relationship indicates that a concept is an attribute of another concept, e.g., an $(ConceptA\ AttributeOf\ ConceptB)$ $ConceptB$ has to be of type Class or of type Object and $ConceptA$ needs to be of type Class. This relationship, also referred to as PropertyOf in some information models, has typically the same semantics as attributes in (object-)relational databases

.

PartOf: This relationship indicates that a concept is a part of another concept, e.g., an edge

($Mitochondrion\ \ PartOf\ \ Cell$) indicates that $Mitochondrion$ is part of a $Cell$. The first concept is of type Class while the second concept can be of type Class or Object. In relational databases, such relationships are often coded as attributes, but we believe that this relationship is sufficiently different semantically from the relationship $AttributeOf$ to warrant separate consideration.

InstanceOf: This relationship indicates that an object is an instance of a class. Therefore, the first concept in the relationship is of type object and the second of type Class. For example, an edge ($LeucineAminopeptidase\ InstanceOf\ Aminopeptidase$) indicates that $LeucineAminopeptidase$ is an instance of the Class $Aminopeptidase$.

ValueOf: This relationship is used to indicate the value of an attribute of an object, e.g., ("49096" $ValueOf\ \ MolecularWeight$). Thus, the first concept is of type literal and the second of type Class. Typically, the second concept (in our example, the class $MolecularWeight$), in turn has an edge (in our example, ($MolecularWeight\ \ AttributeOf\ \ Protein$)) from the object it describes.

Sequences

By itself, the graphical ONION model, described above, does not impose order among the children of a node. In a number of biological applications, say involving DNA sequences, the ability to express order is paramount. In order to express order, we introduce a special relationship, namely $Sequence$ (we refer to a more general notion of sequences including but not only limited to DNA sequences), which is very similar to the container $Sequence$ in RDF. For example, a list ranking can be described using the edges ($MyProtein\ Sequence\ MyProteinSequenceList$), ($MyProteinSequenceList\ :1\ Seq1$), ($MyProteinSequenceList\ :2\ Seq2$). The intermediate node MyProteinSequenceList represents the list object and its elements form an ordered sequence. In an edge of the form ($ConceptA\ Sequence\ ConceptB$) the first concept can be a class or an object and the second concept is an object representing the list. The individual elements of the list can be objects or classes and are related to the list-object via the relationships $:1, :2, \ldots, :N$ where the list has N elements.

The common conceptual model is used to bring ontologies to a common format - so that the articulation generator needs to understand only one format. So if a feature cannot be translated into our common conceptual model, it will not be matched with similar features carrying similar semantic messages in other ontologies. However, such information will still be accessible from the individual ontology and the engine associated with the individual sources.

We resolve the heterogeneity with respect to ontology models and modeling languages by building wrappers that convert ontologies using various conceptual models to an ontology in our common conceptual model. However, the second problem of semantic heterogeneity among the concepts used in the source models still remains. In the next section, we will summarize various methods that we use to automatically suggest ontology articulations.

3. Resolving Semantic Heterogeneity

An important requirement for the application scenarios that our system will be used for is high precision. At this point we believe that resolving semantic heterogeneity entirely automatically is not feasible. We, therefore, advocate a semi-automatic approach wherein an automatic *articulation generator* suggests matches between concepts in the two ontologies it is articulating. A human expert, knowledgeable about the semantics of concepts in both ontologies, validates the generated suggested matches using a GUI tool. An expert can delete a suggested match or say that the match is irrelevant for the application at hand. The expert can also indicate new matches that the articulation generator might have missed. The process of constructing an articulation is an iterative process and after the expert is satisfied with the rules generated, they are stored and used when information needs to be composed from the two ontologies. Figure 1 shows the structure of the ONION system and the interaction between its various components.

In order to keep the cost of computation and especially maintenance (which often dominates other costs in established business environments) low, we strive to make the articulations minimal. Currently, the onus is on the expert to keep the articulation minimal. In future, we hope to make the automated heuristics aware of the needs of the application and minimize the articulations.

The matching algorithms that we use can be classified into two types - iterative and non-iterative.

Non-iterative Algorithms

Non-iterative algorithms are ones that generate the concepts that match in the two ontologies in one pass. These algorithms do not generate any new matches based on existing matches. The non-iterative algorithms that we employ involve matching the nodes based on their content.

The articulation generator looks at the words that appear in the label of the two nodes (or associated with the two nodes, e.g., if the nodes are documents or if more elaborate descriptions of the concepts that are represented using the nodes are available) that it seeks to match and generates a

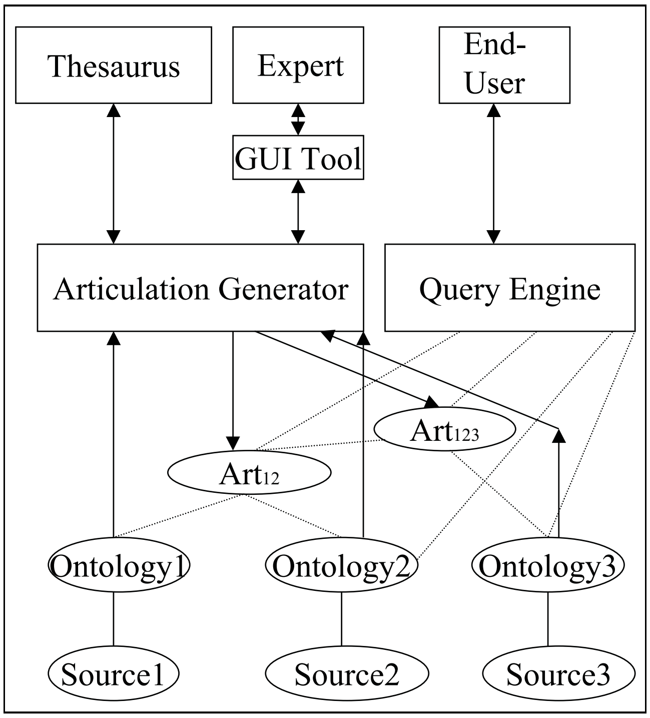

Figure 1. The components of the ONION **system**

measure of the similarity of the nodes depending upon the similarity of the words used in their descriptions or labels.

The non-iterative methods that we currently use primarily refer to dictionaries and thesauri and also use several semantic indexing techniques based on the context of occurrence of words in a corpus. Since the vocabulary used in most bioinformatics applications is highly specialized, the thesauri need to pertain to the specific domain. Similarly, the corpus used in the semantic indexing techniques need to be specific to the application. The quality of the articulation greatly depends upon the quality of the different thesauri and corpora used. The articulation generator is modular in nature, it should be easy to add any other heuristic that allows us to generate semantic similarity measures between terms.

Iterative Algorithms

Iterative algorithms require multiple iterations over the two source ontologies in order to generate semantic matches between them. These algorithms look for structural isomorphism between subgraphs of the ontologies, or use the rules available with the ontologies and any seed rules provided by an expert to generate matches between the ontologies.

For example, one heuristic we use is to look at the attributes of each node and see if the attributes of the two nodes have matched. If a reasonably large number of attributes are the same, the two nodes are related. If all the attributes of one node are also attributes of another node, the articulation generator indicates that the second node is a subclass of the first node. Another heuristic matches nodes based on the matches between their parent (or child) nodes. The expert has the final decision whether to accept this educated guess generated by the articulation generator.

Due to space limitations, we will not describe in detail all the heuristic algorithms that we use to match ontologies, but refer the interested reader to [14].

In the next section, we will briefly define an Ontology Algebra, which allows us to systematically compose information from diverse information sources. Since we focus on small, well-maintained ontologies order to achieve high-precision, but we still want to serve substantial applications, we will often have to combine results of prior articulations. The ontology algebra provides the compositional capability, and thus enhances the scalability of our approach.

4. Ontology Algebra

The key to the scalability of ONION is the systematic and effective composition of information. In this section, we present an algebra that allows us multiple levels of composition of information. By retaining a log of the composition process, we can also, with minimal adaptations, replay the composition whenever any of the sources change [15].

The algebra has one unary operator: $Select$, and three binary operations:$Intersection$, $Union$, and $Difference$.

The $Select$ operator allows us to highlight and select portions of an ontology that are relevant to the task at hand. Given an ontology and a node, the select operator selects the subtree rooted at the node. Given an ontology and a set of nodes, the select operator selects only those edges in the ontology that connect the nodes in the given set.

4.1. Binary Operators

Each binary operator takes as operands two ontologies that we want to articulate, and generates an ontology as a result, using the articulation rules. The articulation rules are generated by an articulation generation function briefly discussed above.

4.1.1. Intersection

Intersection is the most important and interesting binary operation. Let $O1 = (N1, E1, R1)$, and $O2 = (N2, E2, R2)$ be two ontologies. The intersection of two ontologies with respect to AR, a set of articulation rules, generated by an articulation generating function f_{ar} is:
$OI_{1,2} = O1 \cap_{AR} O2 = (NI, EI, RI)$, where,

$NI = Nodes(AR(O1, O2))$,
$EI = Edges(E1, NI \cap N1) + Edges(E2, NI \cap N2) + Edges(Arules(O1, O2))$,
and $RI = Rules(O1, NI \cap N1) + Rules(O2, NI \cap N2) + AR(O1, O2) - Edges(AR(O1, O2))$.

The nodes in the intersection ontology are those nodes that appear in the articulation rules. The edges in the intersection ontology are the edges among the nodes in the intersection ontology that were either present in the source ontologies or have been established as an articulation rule. The rules in the intersection ontology are the articulation rules that have not already been modeled as edges and those rules present in the source ontology that use only concepts that occur in the intersection ontology.

Articulation rules are of two types - ones that are simple statements expressing binary relationships and the more complex rules expressed in Horn Clauses that are mostly supplied by the expert. An example of rules of the former type is: $(O1.Aminopeptidase\ SubclassOf\ O2.Enzyme)$ and one of the more complex logic-based ones is conjunctive rules of the form:
$(O1.X\ InstanceOf\ O1.Endopeptidase)$,
$(O1.CatalyticMechanism\ AttributeOf\ O1.X)$,
$("Unknown"\ ValueOf\ O1.CatalyticMechanism)$
$\Rightarrow (O1.X\ InstanceOf\ EC\_3.4.99)$.
The later establishes the fact that endopeptidases whose catalytic mechanism is not known are classified as EC 3.4.99. The former set of rules are modeled as edges in the articulation ontology and the second set of rules, which require some form of reasoning to derive statements from, are left as rules in the articulation ontology. These rules will be processed during the query evaluation process as necessary.

For all articulation generator functions, we require that $O1 \cap_{ARules} O1 = O1$, that is the articulation generator function should generate such articulation rules that upholds the above-mentioned property as a sanity-check. Articulation generator functions that do not satisfy the above equality are *unsound* and for the purposes of our compositions, we do not use any unsound articualtion generator function.

Note that since we consider each node as an object instead of the subtree rooted at the node, we will get only the node in the intersection by virtue of its appearing in an articulation rule and not automatically include its attributes or subclasses. If the application's query processor requires the attributes of an object in the intersection, it has to get that information from the original source. The label of each node in the intersection maintains the identifier of the source ontology in which the node appears. By not including the entire subtree of a node in the intersection, we reduces its size and thereby its maintenance costs.

As can be seen from the definition of the intersection operator, it is heavily influenced by the articulation rules generated by the articulation generation function.

Theorem 1 *Given two ontologies and the articulation rules $ARules$, the intersection of the two ontologies is unique and can be completely determined.*

Due to space limitations, We will not give the proofs of the lemmas and theorems outlined in the paper but refer the interested reader to [16].

4.1.2. Union

The union $OU = O1 \cup_{ARules} O2$ between two ontologies $O1 = (V1, E1, R1)$ and $O2 = (V2, E2, R2)$ is expressed as $OU = (VU, EU, RU)$, where, $VU = V1 \cup V2 \cup VI_{1,2}$, $EU = E1 \cup E2 \cup EI_{1,2}$,
and $RU = R1 \cup R2 \cup RU_{1,2}$,
and where $OI_{1,2} = O1 \cap_{ARules} O2 = (VI_{1,2}, EI_{1,2}, RI_{1,2})$ is the intersection of the two ontologies.

The union operation combines two source ontologies retaining only one copy of the concepts in the intersection. Though queries are often posed over the union of several information sources, the union of two source ontologies is seldom materialized, since our objective is not to integrate source ontologies but to create minimal articulations and interoperate based on them.

4.1.3. Difference

The difference between two ontologies $O1$ and $O2$, written as $O1 - O2$, includes portions of the first ontology that are not common to the second ontology. The difference can hence be rewritten as $O1 - (O1 \cap_{ARules} O2)$. The nodes, edges and rules that are not in the intersection ontology but are present in the first ontology comprise the difference.

One of the objectives of computing the difference is to optimize the maintenance of articulation rules. An articulation might need to be updated when one of the source ontologies that it articulates is changed.

A change in the source ontology is to be forwarded to the articulation engine. The articulation engine then checks if the changes are confined to the difference between the ontology and the other ontologies that it has been articulated with. If the change happens to be in the difference, then it does not occur in the intersection and is not related to any of the articulation rules that establish semantic bridges between ontologies. Therefore, the articulation rules do not need to be changed. If the changes to a source ontology, instead, is not in the difference, the articulation in which it occurs needs to be updated to reflect the change in the source ontology.

4.2. Properties of the Operators

Optimization of queries often depend upon the ability to rearrange operands, which depends upon the properties of the operators. The properties of the articulation generating function, in turn, determines the properties of the operators.

4.2.1. Commutativity

If an operator is commutative the query optimizer can reverse the order of the operands.

Intersection

Theorem 2 *The intersection is commutative iff the articulation generation function is commutative.*

(Sketch) If an articulation generation function is commutative, the articulation rules generated by it are the same irrespective of the order of the operands. Now from the lemma above, the intersection of two ontologies are entirely determined by the source ontologies and the articulation rules and is unique. Therefore, for the same set of ontologies and articulation rules, the intersection is same. That is, the intersection is commutative, if the corresponding articulation generation function is commutative. The proof of the other direction is proved in [16].

Now, consider the example where we have nodes $O1.Aminopeptidase$ and $O2.Enzyme$. An articulation generator, say $AR1$, can generate two rules based on the order of the operands. For example, $AR1(O1, O2)$ might generate the rule $(O1.Aminopeptidase \ SubClassOf \ O2.Enzyme$, while $AR1(O2, O1)$ might generate the rule $(O2.Enzyme \ SuperClassOf \ O1.Aminopeptidase)$. From the definition of the vocabulary, let us assume that the relationships $SubClassOf$ and $SuperClassOf$ are inverse of one another i.e., $(X \ SubClassOf \ Y) \Leftrightarrow (Y \ SuperClassOf \ X)$ for any X,Y. However, from our strict definitions, the intersection operator with $AR1$ as the articulation generator, is not commutative since the articulation rules - even though are equivalent - are not the same.

This leads us to define the concept of *semantically commutative*.

Definition 1 *An articulation generation function, AR, is* semantically commutative *iff* $AR(O1, O2) \Leftrightarrow AR(O2, O1) \forall O1, O2$*, where* $O1$*, and* $O2$ *are ontologies.*

Definition 2 *An intersection operator is semantically commutative iff the articulation generation function that it uses to derive the articulation rules is semantically commutative.*

To determine the semantic commutativity of articulation generation functions, we need to prove that for any pairs of ontologies, the articulation rules produced by the articulation generation function, even if they are dependent upon the order of the ontologies and are not exactly the same, are in fact equivalent. Automatically proving an articulation generator commutative or semantically commutative is a very computationally intensive job. ONION requires the programmer of the articulation generation function and/or the expert to indicate whether optimization can be enabled by reversing the order of the operands.

For example, articulation generation functions that look up a thesaurus to find synonyms for proteins would generate the same articulation rules (indicating some protein-names refer to the same protein) irrespective of the ontology the protein-name occurs in. Such an articulation generation function would be obviously commutative, making the operator using the function commutative.

Even though the articulation generation function is not strictly commutative, but instead only semantically commutative, the ONION system will take the liberty of reversing the order of the operands to the intersection operator in case such a change gives us performance benefits.

Intuitively, the intersection is entirely determined by the two ontologies and the articulation rules generated by the articulation generation function. If the articulation rules generated are equivalent, then the resulting intersection will also be semantically equivalent and we allow such an optimization in our system.

Union

For the union operator to be commutative the following must hold: $O1 \cup O2 = O2 \cup O1$.

That is, we require $V1 \cup V2 \cup VI_{1,2} = V2 \cup V1 \cup VI_{2,1}$, where $V1$,$V2$, and VIs are the vertices in the source ontologies and the intersection ontologies respectively. The above equality holds only if $VI_{1,2} = VI_{2,1}$. Now the set of nodes in the intersection of two ontologies, VI is obtained from the articulation rules generated from the articulation generator function. $VI_{1,2} = VI_{2,1}$ holds iff $Nodes(ARules(O1, O2)) = Nodes(ARules(O2, O1))$. Similar equalities can be derived from the equations obtained using the edges and rules. We conclude, that as in the case of intersection:

Theorem 3 *The Union operator is (semantically) commutative iff the articulation generator function is (semantically) commutative*

Thus, the conditions for reordering the operands to the union operator are exactly the same as those for the intersection operator.

Difference

Clearly the difference operator is neither commutative nor semantically commutative.

4.2.2. Associativity

If the operators are associative, then the order of their execution can be changed and in order to optimize the execution of the query. The associativity of the relational join operator allows database system designers to rearrange operands and design various optimized join algorithms that improve the performance of queries tremendously. Similarly, the performance of the intersection and union operations can be greatly improved if the operators are associative.

However, before we can state the necessary conditions that makes the intersection operation associative, we need the following definition.

Definition 3 *An articulation generator is said to be* transitively connective *iff for any three ontologies $O1$, $O2$, $O3$, if it generates an articulation rule r_{12} involving $O1.A$ and $O2.B$ while matching $O1$ and $O2$, and another articulation rule r_{23} involving $O2.B$ and $O3.C$ while matching $O2$ and $O3$, then it must generate an articulation rule involving $O1.A$ and some node in $O3$ and also an articulation rule involving $O3.C$ and some node in $O1$ while matching $O1$ and $O3$.*

In other words, if it discovers that $O1.A$ is related to $O2.B$ and $O2.B$ is related to $O3.C$, then it must relate $O1.A$ to some node in $O3$ and $O3.C$ to some node in $O1$.

In the rest of the paper, we use the notation $Rule(n_1, n_2)$ to refer to a rule using the terms n_1 and n_2.

Definition 4 *An articulation generator function f is said to be* consistent *iff* $\forall n_1 \epsilon O1, n_2 \epsilon O2, n_1 \epsilon O3, n_2 \epsilon O4 : Rule(n_1, n_2) \epsilon f(O1, O2) \Leftrightarrow Rule(n_1, n_2) \epsilon f(O3, O4)$ *where $O1$, $O2$ are the source ontologies and $O3$,$O4$ are either source ontologies or intermediate ontologies generated while computing the operations that use f to articulate.*

Loosely speaking, an articulation generation function is consistent if it either always generates or never generates a rule involving two nodes irrespective of the presence or absence of other nodes and edges in its "neighbourhood".

Theorem 4 *The intersection operator using an articulation generation function f that is consistent is associative iff f is transitively connective.*

Note that transitive connectivity is a sufficient condition for the intersection to be associative but is not necessary.

In a lot of situations in practice, the person wishing to compose the two ontologies does not necessarily have a preference in the order in which the ontologies are composed. In such a scenario, the composer might instruct the system to assume associativity (and thus enable rearranging of the operands), even though the articulation generation function is not provably strictly transitively connective (and thus the intersections are not entirely independent upon the order of the operands).

Theorem 5 *The Union operation is associative if the associated articulation generation function f is consistent.*

A consistent articulation generation function f is a sufficient condition for $Union$ with respect to f to be associative. However, it is not necessary for f to be consistent in order for $Union$ to be associative.

Difference is not associative under any conditions.

5. Conclusion

In this paper we present a brief overview of the ONION system that can be used to interoperate among the various information sources available to biologists. ONION uses a simple conceptual model to which different bio-ontologies are mapped using wrappers. The articulation generator is then applied to ontologies expressed using the ONION conceptual model to generate semantic correspondences leading to articulation rules among concepts in the source ontologies. An expert then validates the generated rules or supplies new rules. These rules form the basis of interoperation among the autonomously maintained information sources. Finally, we presented an ontology algebra that provides the formal basis for composition of information and the maintenance of the articulations. The ONION approach supports precise composition of information from multiple diverse sources requiring human-validated articulation rules among such sources. This approach allows the reliable exploitation of information sources that are autonomously maintained without any imposition on the sources themselves. The algebra based on the articulation rules allows systematic composition, which unlike integration is much more scalable. We presented the properties of the algebraic operations, which forms the basis for query optimization while composing information from multiple sources.

References

[1] C. Goble. Carole's position statement. *http://www.cs.man.ac.uk/mig/people/carole/carole.html*, 2000.

[2] Neuroinformatics home page. *http://www.nimh.nih.gov/neuroinformatics/index.cfm*, 2001.

[3] P. D. Karp. A strategy for database interoperation. *Journal of Computational Biology*, 2(4):573–583, 1996.

[4] M. D. Siegel C. H. Goh, S. E. Madnick. Semantic interoperability through context interchange: Representing and reasoning about data conflicts in heterogeneous and autonomous systems http://citeseer.nj.nec.com/191060.html.

[5] C. H. Goh, S. Bressan, S. Madnick, and M. Siegel. Context interchange: new features and formalisms for the intelligent integration of information. *ACM Transactions on Information Systems*, 17(3):270–270, 1999.

[6] The Gene Ontology Consortium. Gene ontology: tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.

[7] R. Stevens. Bio-ontology reference collection. *http://img.cs.man.ac.uk/stevens/onto-publications.html*, 2001.

[8] S. Melnik. Declarative mediation in distributed systems. In *Proceedings of the International Conference on Conceptual Modeling (ER'00)*, 2000.

[9] Unified modeling language: http://www.omg.org/technology/uml/index.htm. 2000.

[10] Daml+oil http://www.daml.org/2001/03/daml+oil-index. 2001.

[11] M. Gyssens, J. Paredaens, and D. Van Gucht. A graph-oriented object database model. In *Proc. PODS*, pages 417–424, 1990.

[12] P. Mitra, M. Kersten, and G. Wiederhold. A graph-oriented model for articulation of ontology interdependencies. In *Advances in Database Technology-EDBT 2000*, Lecture Notes in Computer Science, 1777, pages 86–100. Springer-Verlag, 2000.

[13] Resource description framework(rdf) model and syntax specification, w3c recommendation http://www.w3.org/tr/rec-rdf-syntax. 1999.

[14] P. Mitra, G. Wiederhold, and J. Jannink. Semi-automatic integration of knowledge sources. In *Proc. of the 2nd Int. Conf. On Information FUSION'99*, 1999.

[15] J. Jannink. *A Word Nexus for Systematic Interoperation of Semantically Heterogeneous Data Sources*. PhD thesis, Stanford University, 2000.

[16] P. Mitra. An algebra for semantic interoperation of information sources, http://www-db.stanford.edu/ prasen9/alg.txt. Technical report, Infolab, Stanford University, July 2001.