# A Scalable Framework for the Interoperation of Information Sources

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

**Abstract.** Resolving heterogeneity among information systems is a crucial necessity if we wish to gain value from the many distributed resources available to us. Problems of heterogeneity in hardware, operating systems, and data structures have been widely addressed, but issues of diverse semantics have been handled mainly in an ad-hoc fashion. In this paper, we present ONION, a system based on a scalable approach to interoperation of information systems that articulates their associated ontologies. An articulation focuses on the semantically relevant intersection of information resources with respect to a type of application. However, ontologies obtained from diverse sources are represented using different formats. We have designed a simple intermediate format - the ONION ontology format - that we transform ontologies to before we generate semantic correspondences or articulations between them.

In ONION, application-dependent articulation rules that capture the correspondence between concepts in different ontologies are established between source ontologies semi-automatically. Finally we present an ontology algebra, based on the articulation rules, for the composition of ontologies.

## 1 Introduction

Today a large number of diverse information sources - databases, knowledge bases, collections of documents - are available on the Internet. Often, we cannot answer a query by using a single source, and need to compose knowledge from multiple sources. Intelligent searching and querying on the World Wide Web - the largest collection of distributed information and knowledge sources - often requires composing information from heterogeneous information sources. Today, the bulk of this composition is done by the end-user. Not only is this process extremely tedious and time-consuming, but also, often, the end-user does not have any idea about the semantics used by the builder of the information source. In this paper, we present a brief overview of the ONION (ONtology compositION) system, which takes a systematic approach to enable semi-automatic interoperation among heterogeneous information sources.

### *1.1 Heterogeneity*

Most information sources have been independently constructed and are autonomously maintained. Attempts have been made to integrate information from these various information

* This work was partially supported by a grant from the Air Force Office of Scientific Research (AFOSR).

sources into a monolithic information source [1], [2]. Such an approach, however, creates maintenance and scalability problems. When an information source is to be added, the large information source must be restructured, and often, this kind of maintenance leads to substantial delays [3].

Some researchers have tried to first build a standard ontology or global schema and then build information sources that conform to that ontology or schema [4], [5]. Even though this approach has worked for small communities, coming up with an agreed-to-standard for knowledge in larger domains is almost impossible, especially among groups that have different applications in mind.

Besides, it is prohibitively expensive to restructure existing knowledge so that it conforms to the standard ontology even if one were able to.

### *1.2 Maintenance*

Everyday new discoveries expand our knowledge and change our views of the universe that we live in. Therefore, even if information sources start off with a common ontology, this ontology has to be updated periodically. The maintainers of the information sources that use the standard ontology will have to agree on the updates being proposed and on the restructuring of the ontology. They may have entirely different applications in mind or may not subscribe to a newly discovered theory. Furthermore, some participants might see the changes required to support the proposed updates as an unnecessary imposition since restructuring the information source will require substantial effort on their part. Thus generating new consensus on updates to the standard ontology is a time-consuming and tenuous process. For quickly changing fields, arriving at a consensus within a short period of time is not even feasible. Therefore, we need a system where the information sources are autonomous.

### *1.3 A Realistic Setting*

We believe that the information sources should be autonomous and that they should not be required to conform to a standard ontology in order for a user to get a composition of knowledge from them. Instead of integrating information sources, we intend to enable interoperation among them.

Unfortunately, the composition of knowledge from multiple independently maintained information sources is a hard problem. Independently constructed information sources are heterogeneous and often use different vocabularies and data formats. The organization of class-subclass hierarchies are substantially different. Often, they use different terms to represent the same concept and the same term to represent entirely different concepts. In order to interoperate among such information sources we need to resolve their semantic heterogeneity.

Karp [6] proposes a strategy for database interoperation. We extend Karp's approach to apply to not only databases, but also to knowledge bases and information sources.

As in [7], [8], and [6], we assume that information sources are independently created and maintained. In Karp's system, each database comes with a schem a which is saved in a Knowledge Base of Databases. 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.

The ontologies associated with information sources are based on some existing, known vocabularies and data formats. Native drivers and wrappers provide access to the ontol ogies and help us restructure the information if needed. We establish application-specific *articulation rules*, i.e., rules that establish correspondence between concepts in different ontologies, semi-automatically.

Queries are rewritten using the articulation rules. Before a query is dispatched to a source, the terms in the query are rewritten using the articulation rules that indicate the semantic correspondence between the terms in the query and those in the source. This rewriting ensures that a source gets a query that conforms to the vocabulary and the semantics of the source. During query planning, optimization is enabled based on the algebraic properties of the operations.

In this paper, we describe the ONION system and highlight our approach to interoperation. 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 Composition Algebra that we use to compose ontologies. Section 5 concludes the paper.

## 2 The ONION Ontology Format

ONION's articulation generator needs all source ontologies to conform to a common ontology format. Heterogeneity among information sources needs to be resolved 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 data formats and modeling languages to represent their data and meta-data. In the rest of the paper, we will refer to the data format and the modeling language together as the ontology format. Second, sources using the same data format differ in their semantics. The ONION system uses a common ontology format, which we describe below. It first converts all external ontologies to this common format and then resolves the semantic heterogeneity among the objects in the ontologies that it is articulating.

Melnik, et al., [9] have shown how to convert documents from one modeling format to another, e.g., from RDFS [10] to DAML+OIL [11]. We can use wrappers to convert the ontologies represented using the formats that we want to support to the ONION format.

Instead of writing a wrapper to convert all ontologies from their native formats to the ONION format, we could do so declaratively. That is, we could first write rules that transform parts of one ontology from one format to another. However, such an approach is not feasible. Often, the transformations are quite complex and can be more naturally expressed procedurally. Expressing them declaratively requires a very expressive rule language. Deductions in more expressive rule languages are often not tractable. Also, we would have to create and manipulate articulation rules that not only have semantic information but also have information about transforming ontology formats. Besides, by converting to the ONION format, we eliminate the necessity of $n^2$ pariwise conversions among $n$ ontologies and instead reduce it to $n$ conversions (of all the ontologies to the common format).

We solve the problem of establishing correspondences among ontology formats and the problem of establishing articulations among the concepts in the ontologies differently because we believe that the small number of ontology formats that we intend to support (currently XML, RDF, DAML+OIL) can be converted to use one common format, whereas the number of concepts and thus objects used in ontologies are rather large and creating a huge, integrated, common, global ontology is untenable and if such an ontology could be created it would be unmaintainable.

Information sources were, are and will be modeled using different formats. We do not foresee the creation of a *de facto* standard data format 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 which the source ontologies are converted. ONION then matches the converted ontologies to create the articulation ontology. The common ontology format could be arbitrarily complex so that we could transform all features from various ontology formats minimally. However, to make the articulation generation simple, we take a different approach and strive to keep our model simple.

### *2.1 A Graph-Oriented Conceptual Model*

Our common ontology format is based on the work done by Gyssens, et al.,[12]. At 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 $E$. $R$ is expressed as Horn clauses.

An edge *e* is written as $(n_1, \alpha, n_2)$, where $n_1$ and $n_2$ are two nodes belonging to the set of nodes $V$ and $\alpha$ is the label of the edge between them. The label of a node $n$ is given by a function $\lambda(n)$ that maps the node to a non-null string. In the context of ontologies, the label is often a noun-phrase that represents a concept. The label $\alpha$ of an edge $e = (n_1, \alpha, n_2)$ is a string given by $\alpha = \delta(e)$. The label of an edge is the name of a semantic relationship among the concepts and it can be null if the relationship is not known. The domain of the functions $\lambda$ and $\delta$ is the universal set of all nodes and edges, respectively (from all graphs), and their range is the set of strings (from all lexicons). For the rest of the paper, we assume that the function $\lambda$ maps a node to a unique label (i.e., the name of the node in the ontology is concatenated with the name of the ontology). Thus, we will use the label of a node as a unique identifier of the node. To represent an edge, we can substitute the label of a node for a node and write edge $e = (\lambda(n_1), \alpha, \lambda(n_2)$.

The graph in the ONION data format can be expressed using RDF [13]. Each edge in our graph is coded as an RDF sentence, with the two nodes in the edge 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 for RDF and do not have to write parsers and other tools for our model.

Rules in an ontology are 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, in order to limit the computational complexity we will use a simpler language like Horn Clauses. A typical rule $r \in R$ is of the form :

$$CompoundStatement \implies Statement$$

A *Statement* is of the form $(Concept\ Relationship\ Concept)$. A $Concept$ is either a label of a node or a variable that can be bound to one or more nodes in the ontology graph. A $Retationship$ expresses a realtion between two $Concept$s.

### 2.2 *Semantic Relationships in* ONION

The ONION *articulation generator* can improve the semantic matches it derives among concepts in a pair of ontologies if it has some semantic information about the relationships used in the ONION ontology model.

Certain data formats 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. A system that knows the exact semantics of the relationships in a conceptual model can use the information, for instance, to find better matches between concepts in two ontologies or to perform type-checking and flag errors.

Other models allow any user-defined relationships, without any restriction. For instance, relationships like $OwnerOf$ tend to be interpreted according to the semantics associated with them by the local application. Such relationships need not be strictly typed, and a general system that imports a model allowing them does not know of the application-specific semantic interpretation of the relationships. This approach provides enormous flexibility and can accommodate a large number of relationships. However, since the semantics of these relationships are not exactly known by the system, it cannot use them for matching related concepts or for type-checking.

The ONION data formating encourages the use of a set of strictly-typed relationships with precisely defined semantics. The set of relationships that our articulation generator knows the semantics of is $\{SubClassOf, PartOf, AttributeOf, InstanceOf, ValueOf\}$.

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 we have chosen a different and much smaller set of relationships to define semantics in ONION so as to maintain its simplicity.

In the following description, we use the term $Literal$ to denote a string and the terms $Class$ and $Object$ to denote classes and objects in the sense it is used in object-oriented databases. The semantics of the set of pre-defined relationships available in our common ontology format are as follows:

$SubClassOf$: relates two concepts each of type $Class$. The relationship indicates that one concept is a subclass of another. For example, the statement $(Car\ SubClassOf\ Vehicle)$ denotes that the concept $Car$ is a subclass of concept $Vehicle$. Any instance of the class $Car$ is also an instance of the class $Vehicle$ and all the attributes of the class $Vehicle$ are also attributes of the class $Car$. The relationship $SubClassOf$ is transitive, and in the absence of an explicit rule in an ontology that states that the $SubClassOf$ relationship is transitive, we will add one to the ontology before reasoning or rewriting the queries using the rules.

$AttributeOf$: This relationship indicates that a concept is an attribute of another concept, e.g., an edge $(ConceptA\ AttributeOf\ ConceptB)$ indicates that $ConceptA$ is an attribute of $ConceptB$. $ConceptB$ is of type $Class$ or of type Object and $ConceptA$ is 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 . The following rule holds:

$$((A\ AttributeOf\ B)\ \wedge\ (B\ SubClassOf\ C))\implies(A\ AttributeOf\ C)$$

$PartOf$: This relationship indicates that a concept is a part of another concept, e.g., an edge $(Chassis\ PartOf\ Car)$ indicates that $Chassis$ is part of a $Car$. 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 $(MyCar\ InstanceOf\ Car)$ indicates that $MyCar$ is an instance of the Class $Car$. The following rule holds:

$$((A\ InstanceOf\ B)\ \wedge\ (B\ SubClassOf C))\ \Longrightarrow\ (A\ InstanceOf\ C))$$

$ValueOf$: This relationship is used to indicate the value of an attribute of an object, e.g., ("29" $ValueOf\ Age$). Thus, the first concept is of type $Literal$ and the second of type $Class$. Typically, the second concept (in our example, the class $Age$), in turn, has an edge (in our example $(Age\ AttributeOf\ PersonA)$) from the object it describes.

### *2.3 Sequences*

XML is becoming a popular format for expressing data and meta-data on the web. Like SGML and other markup languages primarily designed to express documents, XML imposes order among its elements. By itself, the graphical ONION model, described above, does not impose order among the children of a node. To express order, we introduce a special relationship, namely $Sequence$, which is very similar to the container $Sequence$ in RDF. For example, a list ranking cars can be described using the edges

$(MoneyLineRanking\ Sequence\ CarRankingList)$,
$(CarRankingList\ :1\ HondaAccord)$, and
$(CarRankingList\ :2\ FordTaurus)$.

The intermediate node $CarRankingList$ 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.

In the ONION format, we do not require that every relationship must belong to the small set of relationships whose semantics are predefined. The model is flexible enough to allow any other user-defined relationship. The articulation generator will not be able to use the relationships, whose semantics it is not aware of, unless the semantics are captured using rules in the source ontology. For example, if the source ontology uses a relationship $Is-A$ and has a rule that says that "Is-A" is transitive, the articulation generator can use that information to generate matches. 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 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 generator uses only relationships whose semantics are clearly defined to it to derive meaningful matches among the nodes and ignores the relationships that it does not know the semantics of. Therefore, if two RDF models have the relationships "Buyer" and

”Owner” and for the purposes of the application we want to generate a match between the two, we need to represent these relationships as nodes in the ONION model and then run the articulation generator to match them.

### 2.4 Reference and Subsumption

In data formats, especially those used to model documents, like XML, SGML, OEM etc. [14], where there are nested objects and entities, an object is modeled as a subtree in a graph. The entire subtree rooted at a node comprises the object that the node represents. When a query asks for the object, the entire subtree is returned. Such models assume that an object subsumes all objects that are in its subtree. If any relationship needs to be expressed between two objects a reference to the second object is used. The reference is denoted by having a node with the the identifier of the second object and having an edge to this node. The use of this additional node that refers to a different object helps preserve the tree structure of the models, which is required for documents, since they are in essence serialized entities.

In our model, however, even though many of the relationships, with pre-defined semantics, are essentially subsumptive in nature, we intend to keep the concept of an object as simple as possible. Faced with the question of defining the scope of an object in our common data format, we take the minimal approach. In our world, a single node represents a concept: a class, an object, or a value. All edges are referential in nature. Thus, when a query asks to select an object, only the node representing the object is returned and not the entire subtree rooted at the node. This minimal definition of an object helps us keep the articulation rules and the resulting ontology intersections as small as possible. As we will see later, the larger the intersection, the greater the cost when using the articulation to answer queries. Thus we make the choice to keep the definition of an object as simple as possible.

Apart from the graph model, our data format allows us to declaratively supply rules. Some features in other models can be converted using the rules to capture their semantics. If this is not possible, relationships which are not interpreted by ONION can be used. Some features still cannot be expressed using the ONION model.

The common data format 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 data format, 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 data format. 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. In distinction to research tasks, casual browsing, and web-surfing, the cost of eliminating false hits is very high in business environments. 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.

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 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 the Nexus [15] and also use several semantic indexing techniques based on the context of occurrence of words in a corpus. Since the articulation generator is modular in nature, it should be easy to add any other sophisticated heuristic (like consulting WordNet [16]) that allows us to generate semantic similarity measures between phrases.

*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. Iterative algorithms are typically used after the non-iterative algorithms have already generated some semantic matches between the ontologies and use these generated matches as its base.

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 bless 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 [17].

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 in 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 Composition Algebra

In this section, we present an algebra for Ontology Composition that allows us to compose information systematically. Depending upon the properties of the algebraic operators, we optimize the composition of ontologies.

By retaining a log of the articulation and subsequent composition process, we can also, with minimal adaptations, replay the composition whenever any of the sources change[15]. Without such a capability, integrated ontologies soon became stale and useless. Redoing a substantial integration manually is rare, because of the cost, and the realization that the work will be obsolete again in a short time.

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

### *4.1 Select*

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

### *4.2 Intersection*

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.

Intersection is the most important and interesting binary operation. The intersection of two ontologies $O1 = (N1, E1, R1)$, and $O2 = (N2, E2, R2)$ with respect to the set of articulation rule generating f unction $AR$ is:
$OI_{1,2} = O1 \cap_{AR} O2$, where $OI_{1,2} = (NI, EI, RI)$,
$NI = Nodes(AR(O1, O2))$,
$EI = Edges(E1, NI \cap N1) + Edges(E2, NI \cap N2) + Edges(Arules(O1, O2))$ ,
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

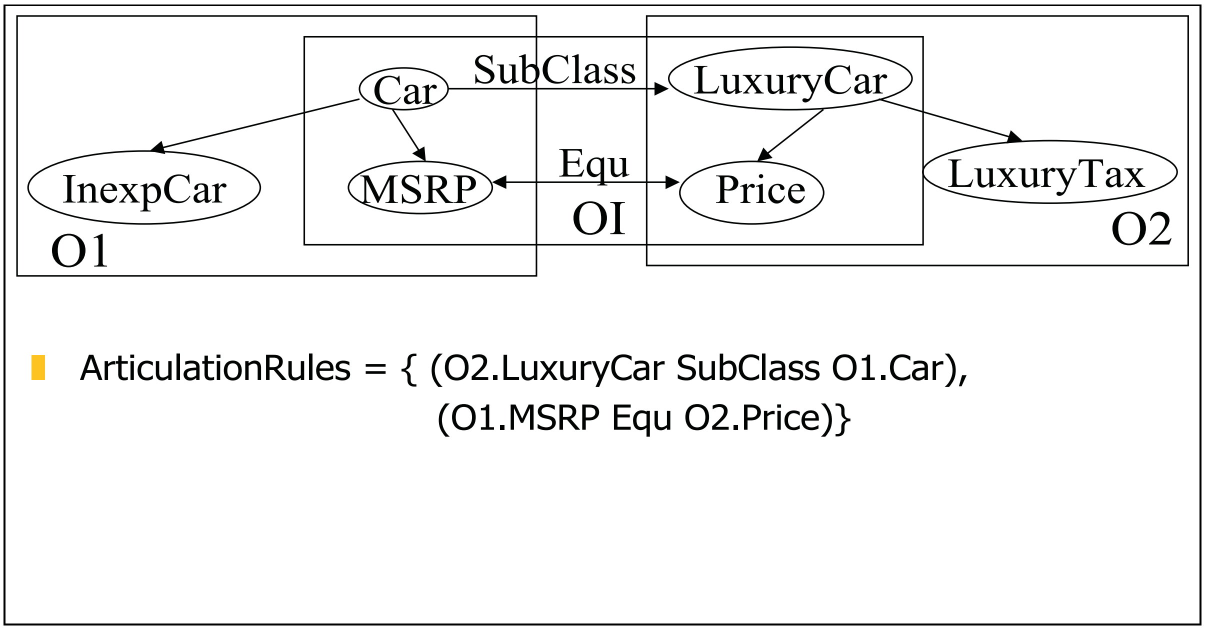


Figure 1: The Intersection Ontology $OI$ of Source Ontologies $O1$ and $O2$

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.

The 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: $(O1.CarSubclassOfO2.Vehicle)$ and one of the more complex logic-based ones is a conjunctive rule of the form: e.g. conjunctive rules of the form $(O1.XInstanceOfO1.Car), (YPriceOfX), (Y > 30000) \Rightarrow (O1.XSubClassOfO2.LuxuryCar)$. 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 belonging to the articulation ontology. These rules will be processed during the query evaluation process only when necessary.

For all articulation generator functions, we require that $O1 \cap_{AR} O1 = O1$, 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.

In Figure 1, we show two ontologies $O1$, $O2$, the articulation rules between them and the intersection ontology $OI$. $Equ$ is a short-hand that we use when to indicate classes that are equivalent in the two ontologies.

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. Again, a minimal linkage serves our needs better than inclusion of possibly irrelevant concepts. Inclusion of attributes will be required to define subclass relationships among nodes in the source ontologies precisely.

Each node in the intersection has a label which contains the URI of the source in which it appears. If the attributes of the object that it represents are required, the application's query processor has to get that information from the original source. Defining the intersection with a minimal outlook reduces the complexity of the composition task, and the maintenance costs, which all depend upon the size of the articulation.

### *4.3 Union*

The union $OU$ between two ontologies $O1 = (V1, E1, R1)$ and $O2 = (V2, E2, R2)$ is expressed as $OU = O1 \cup_{AR} O2 = (VU, EU, RU)$ 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 $OI_{1,2} = O1 \cap_{AR} O2 = (VI_{1,2}, EI_{1,2}, RI_{1,2})$ 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, we expect this operation to be rarely applied to entire source ontologies. 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. However, we do expect that larger applications will often have to combine multiple articulations and here is where the union operation is handy.

### *4.4 Difference*

The difference between two ontologies $O1$ and $O2$ , written as $O1 - O2$, between two ontologies $O1 = (V1, E1, R1)$ and $O2 = (V2, E2, R2)$ is expressed as $OD = (VD, ED, RD)$, where $VD = V1 - VI_{1,2}$, $ED = E1 - EI_{1,2}$, and $RD = RI - RI_{1,2}$, 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. That is, the difference ontology includes portions of the first ontology that are not common to the second ontology. The nodes, edges and rules that are not in the intersection ontology but are present in the first ontology comprise the difference.

### *4.5 Properties of the Operators*

We defined the operators in the algebra on the basis of the articulation rules produced by the articulation generating function. Not surprisingly, most of the properties of the binary operations are based on the properties of the articulation generating function.

For example, the intersection and union operators are commutative if and only if the articulation generation function, on which they are based, is commutative. The commutativity of intersection and union gives the query optimizer the freedom to swap the order of the operands for these operations if required to optimize performance of the query. However, strict commutativity of the articulation generation function might not be achievable or necessary in order to allow the operands to be swapped.

Consider the example where an articulation generator generates articulation rules

$$AR(O1, O2) = (O1.CarNM : SubClassOf O2.Vehicle)$$

and

$$AR(O2, O1) = (O2.VehicleNM : SuperClassOf" O1.Car")$$

Although the articulation generation function is not commutative, the semantic information contained in the articulation rules are equivalent as long as the relations $SubClassOf$ and $SuperClassOf$ defined in the namespace "NM" are semantically similar after we invert their parameters. Thus, if the rules obtained by swapping the operands are semantically equivalent, we can swap the operands without compromising on the correctness of the result.

To capture this, we define the concept of *semantic commutativity*.

**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.*

and the necessary and sufficient condition for intersection to be semantically commutative is:

**Theorem 1.** *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 are in fact semantically equivalent. Automatically proving an articulation generator commutative or semantically commutative might be easy for the $SubClassOf$ and $SuperClassOf$ examples, but is not always feasible. In such cases, ONION requires the programmer of the articulation generation function and/or the expert to indicate whether the function is semantically commutative. In the absence of such information, ONION conservatively assumes that the operation is not semantically commutative if it cannot prove otherwise.

For detailed discussions and proofs of the theorems regarding the necessary and sufficient conditions for the operators to have properties (like commutativity, and associativity), please refer to [18]. We have identified the desired properties that a "well-behaved" articulation generation function should have so that query optimization can be enabled.

Using a formal process minimizes the maintenance costs in two ways: first of all we can recognize when a change in a source does not require a change in the articulation rules, and if a change is required we can rapidly regenerate the affected articulations, and adapt them to the new situation.

## 5 Conclusion

In this paper we present a brief overview of the ONION system used for the interoperation of information sources. ONION uses a simple data format to which different ontology models are mapped using wrappers. The articulation generator is then applied to ontologies expressed using the sc ONION data format to generate semantic correspondences leading to articulation rules among concepts in the source ontologies. A domain expert validates the generated rules or supplies new rules. These rules form the basis of interoperation among the autonomously maintained information sources. Finally, we briefly highlighted 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 by not relying on simple lexical matches, but 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. When sources change maintenance is rapid since the effect of the changes can be determined using the algebra and the composition can be regenerated where needed.

## References


[1] Cia factbook: http://www.cia.gov/cia/publications/factbook/. 2000.

[2] O. Ritter, P. Kocab, M. Senger, D. Wolf, and S. Suhai. Prototype implementation of the integrated genomic database. *Computers and Biomedical Research*, 27:97–115, 1994.

[3] Diane E. Oliver. *Change Management and Synchronization of Local and Shared Versions of a Controlled Vocabulary*. PhD thesis, Stanford University, 2000.

[4] Information integration using infomaster, http://infomaster.stanford.edu/infomaster-info.html.

[5] Thomas Kirk, Alon Y. Levy, Yehoshua Sagiv, and Divesh Srivastava. The information manifold. In C. Knoblock and A. Levy, editors, *Information Gathering from Heterogeneous, Distributed Environments*, Stanford University, Stanford, California, 1995.

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

[7] 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.

[8] 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.

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

[10] Resource description framework(rdf) schema specification 1.0, w3c recommendation http://www.w3.org/tr/rdf-schema. Technical report, 2000.

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

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

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

[14] Serge Abiteboul, Sophie Cluet, and Tova Milo. Correspondence and translation for heterogeneous data. *To appear in Theoretical Computer Science http://osage.inria.fr/verso/PUBLI/all-bykey.php?mytexte=abiteboul*, 2001.

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

[16] Wordnet - a lexical database for english. http://www.cogsci.princeton.edu/ wn/. Technical report, Princeton University.

[17] 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.

[18] 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.