# A Graph-Oriented Model for Articulation of Ontology Interdependencies*


Prasenjit Mitra, Gio Wiederhold
Stanford University
Stanford, CA, 94305, U.S.A.
{mitra, gio}@db.stanford.edu

Martin L. Kersten
INS, CWI, Kruislaan 413
109 GB Amsterdam, The Netherlands
mk@cwi.nl



**Abstract:** *Ontologies* are knowledge structures to explicate the contents, essential properties, and relationships between terms in a knowledge source. Many sources are now accessible with associated ontologies. Most prior work on use of ontologies relies on the construction of a single global ontology covering all sources. Such an approach is not scalable and maintainable especially when the sources change frequently. We propose a scalable and easily maintainable approach based on the interoperation of ontologies. To handle user queries crossing the boundaries of the underlying information systems, the interoperation between the ontologies should be precisely defined. Our approach is to use rules that cross the semantic gap by creating an *articulation* or linkage between the systems. The rules are generated using a semi-automatic articulation tool with the help of a domain expert. To make the ontologies amenable for automatic composition based on the accumulated knowledge rules, we represent them using a graph-oriented model extended with a small algebraic operator set. ONION, a user-friendly toolkit, aids the experts in bridging the semantic gap in real-life settings. Our framework provides a sound foundation to simplify the work of do-



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

main experts, enables integration with public semantic dictionaries, like Wordnet, and will derive ODMG-compliant mediators automatically.

*Keywords:* semantic interoperation, ontology algebra, graph-based model

# 1 Introduction

Bridging the semantic gap to answer end-user queries in a heterogeneous environment is a prerequisite and key challenge to support global information systems. The basis for this bridge is found in an *ontology* for the knowledge sources involved. An *ontology* in this context is defined as a knowledge structure to enable sharing and reuse of knowledge by specifying the terms and relationships among them. Ontologies relate to knowledge sources like dictionaries relate to literary works. Like the dictionary, the ontology collects and organizes the terms of reference. By analogy of the definitions in a dictionary, which give us the relationships between words, ontologies give us the relationships between terms. The ontologies considered are *consistent*, that is, a term in a ontology does not refer to different concepts within one knowledge base. A consistent vocabulary is needed for unambiguous querying and unifying information from multiple sources.

Automation of access to broad information resources, as on the world-wide web, requires more precision than is now afforded by human focussed browsing mechanisms. The predominant route is to rely on XML[1] as a carrier of semantic information. By itself, this is not enough. An XML document can be used to represent a single ontology at best. We focus on such structured worlds as a starting point, establish how to build semantic bridges between them and use reasoning based on such semantic information to compose knowledge from multiple knowledge sources.

Bridging the gap between multiple information sources has long been an active area of research. Previous work on information integration[2] and on schema integration[3] has been based on the construction of a unified schema. However, unification of schemas does not scale well. Not only does it lead to a huge and difficult to maintain schema, but it also, very quickly, calls for identification of 'sub-domains' and 'namespaces'[4] to make the semantic context of distinct sources explicit. Instead, we propose to utilize semantic bridges between such contexts as a starting point so that the source ontologies remain independent. Moreover, in most real-life situations, it suffices to just provide the semantic bridges where interaction is required.

Recent progress in automated support for mediated systems, using materialized views, has been described by [5], [6], [7], and [8]. We share the underlying assumption that a view is a syntactic representation of a semantic context of an information source. Definition of such views, however, requires manual specification. They need to be updated or reconstructed even for small changes to the individual sources. Our contention is that, in many cases, a application domain-specific ontology articulation rule set will simplify the work involved. Such rule sets are implicitly found in the standardization efforts encountered in business chains to enable electronic interaction.

Semantic interoperatibility has been studied in work on heterogeneous databases[9],[10] and multidatabase systems[11], [12]. One of the strategies used is to merge each system schema into a reference schema. However, such a strategy suffers the same drawbacks as the information integration approach. Relying on the end-user to bridge the semantic gap using a multi-database query imposes the implicit assumption that all end-users are domain experts. Instead, we envision a system to propose and manually resolve the semantic gaps between the knowledge sources. Such a system is driven by a rule set supplied by the domain expert. This tedious task is greatly simplified by using a tool that uses external knowledge sources to propose relevant semantic bridges. It also allows the domain expert to provide immediate feedback on potentially ambiguous constructs.

Ontologies have been represented using various text-based models[13]. While a text-based model is easy to construct initially, its lack of structural relationships is a major inadequacy. This becomes especially crucial if the ontologies have to be presented to a human expert or end-user.

We adopt a graph-based model to represent ontologies. A graph-based model conveys the structural relationships in an ontology in a simple, clean, elegant and more usable format. The graphical scheme deployed is a refinement of the GOOD[14] model, which has been developed to model an object-oriented DBMS using a graph-based framework.

In this paper, we show how ontologies of individual knowledge sources can be articulated into a unified ontology using a graphical representation, where semantic bridges are modeled using both logical rules (e.g., semantic implication between terms across ontologies) and functional rules (e.g. dealing with conversion functions between terms across ontologies).

The innovation of ONION (ONtology compositION) system is an architecture based on a sound formalism to support a scalable framework for ontology integration. The architecture provides a balance between an automated (and perhaps unreliable system), and a manual system specified totally by a domain expert. It is based on a modular framework that allows for integration of its other knowledge processing components. The model is simple, yet rich enough to provide a basis for the logical inference necessary for knowlegde composition and for the detection of errors in the articulation rules. An ontology algebra is defined, which is the machinery to support the ontology matching process and to realize its implementation.

This paper is further organized as follows. In Section 2 we give an architectural overview of ONION. In Section 3 we outline the graphical representation of ontologies. Section 4 deals with the generation of the articulation. In Section 5 we introduce an ontology algebra. Finally, in section 6 we summarize the contributions of the paper.

## 2 Overview of ONION

In this section we present an overview of the system architecture and introduce our running example. The ONION architecture, shown in Figure 1, has been designed with strong modularity in mind.

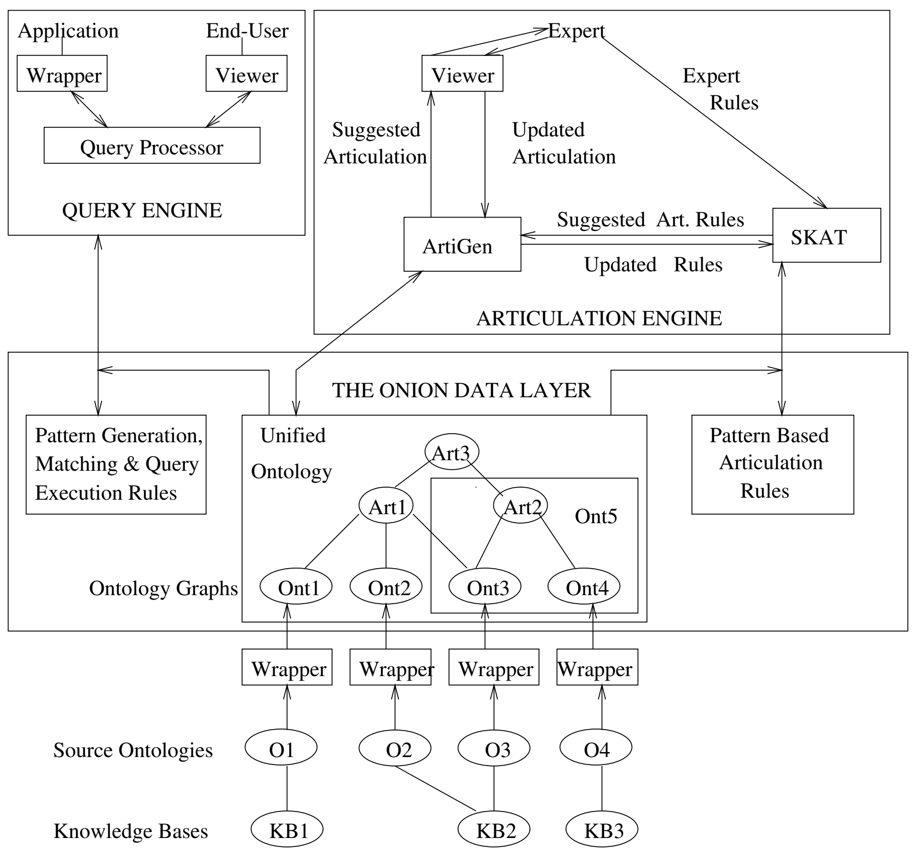


Figure 1: The ONION system

It recognizes the need for several underlying knowledge source representations, supporting different kinds of semantic reasoning components, and integration with query processing engines. By keeping the model simple and the architecture modular, we hope to achieve greater scalability and incur less problems in maintenance.

## 2.1 The ONION data layer

The ONION data layer is the primary focus of this paper. It manages the ontology representations, the articulations and the rule sets involved and the rules required for query processing. Each ontology, $O_i$, reflects (part of) an external knowledge source. Since ontologies may be represented in a variety of ways, we expect some transformations to make them amenable for management within the context of our system. We accept ontologies based on IDL specifications and XML-based documents, as well as simple adjacency list representations.

Most ontology toolkits have inference mechanisms of varying complexities tightly coupled with the ontologies[15]. Our approach is to separate the logical inference engine from the representation model for the ontologies. This allows us to accomodate different inferences engines which can deal with ontology graphs.

The articulations are represented as ontology structures $A_j$ linked with their underlying sources, i.e., the semantic bridges. The articulation ontology commonly uses concepts and structures inherited from the sources. As such, they can be seen as a new knowledge source for upper layers.

The semantic bridges can be cast as articulation rules $R_k$, which take a term from $O_i$ and maps it into a term of $O_j$ using a semantic meaningful label.

## 2.2 The ONION viewer

The ONION viewer is a graphical user interface for the ONION system. A domain expert initiates a session by calling into view the ontologies of interest. Then he can opt for a refinement of an existing ontology using off-line information, import additional ontologies into the system, drop an ontology from further consideration and specify articulation rules.

The alternative method is to call upon the articulation engine to visualize possible semantic bridges based on the rule set already available. The expert can then update the suggested bridges using the viewer or supply new rules for the generation of the articulation. Effectiveness of this method depends on the inference performed by the articulation engine and heuristics embedded in the existing rule set. Progress in capturing business semantics in products such as Business Objects, SAP/R3 and Baan, indicate that such rule sets can indeed be derived.

Once finished, the expert may use the interface to formulate queries or direct the system to generate wrappers for inclusion in concrete applications using the ONION query engine.

## 2.3 The ONION query system

Interoperation of ontologies forms the basis for querying their semantically meaningful intersection or for exchanging information between the underlying sources.

The former calls for a traditional query engine, which takes a query phrased in terms of an articulation ontology and derives an execution plan against the sources involved. Given the semantic bridges, however, query reformulation is often required.

The query system has a graphical user interface, wrappers for applications and a query processor which uses the query execution rules to reformulate the query and generate its solution.

## 2.4 The Articulation Engine

The articulation engine is responsible for generating potential articulations between the source ontologies. ONION is based on the SKAT (Semantic Knowledge Articulation Tool) system developed in recent years at Stanford[16]. Articulation rules are proposed by SKAT using expert rules and other external knowledge sources or semantic lexicons (e.g., Wordnet) and verified by the expert. It uses an inference engine to derive possible alternative articulations.

The articulation generator takes the articulation rules generated by SKAT and generates the articulation. This is then forwarded to the expert for confirmation. If the expert suggests changes or new rules, they are forwarded to SKAT for further generation of new articulation rules. This process is iteratively repeated until the expert is satisfied with the generated articulation.

### Notational conventions

In the rest of the paper we will use the following terms. The individual ontologies will be referred to as *source ontologies*. Articulation rules indicate which terms, individually or in conjunction, are related in the source ontologies. An *articulation ontology* contains these terms and the relationships between them. The term *articulation* will refer to the articulation ontology and the the rules that relate terms between the articulation ontology and the source ontologies. The source ontologies along with the articulation is referred to as the *unified ontology*.

It is important to note that the unified ontology is not a physical entity but is merely a term coined to facilitate the current discourse. The source ontologies are independently maintained and the articulation is the only thing that is physically stored. As shown in, Figure 1, the unified ontology O5 is constructed from the source ontologies O3 and $O_4$ and the articulation ontology $A_2$. The articulation consists of $A_2$ and the rules linking it to $O_3$ and $O_4$.

### 2.5 Motivating Example

To illustrate our graph model of ontologies and articulation, we use selected portions of two ontologies. The portions of the ontologies *carrier* and $factory$ related to a transportation application have been selected (and greatly simplified) (Figure 2). These ontologies model the semantic relationships 'SubclassOf', 'AttributeOf', 'InstanceOf' and 'Semantic Implication' that are represented as edge labels 'S','A','I','SI' respectively. For clarity a few of the most obvious edges have been omitted.

## 3 Graphical Representation of Ontologies

The ONION system has been anchored in the seminal work on graph-based databases in[14]. In this section we introduce its formal setting, the operations that come with it, and how to deploy the graph-based setting in the running example.

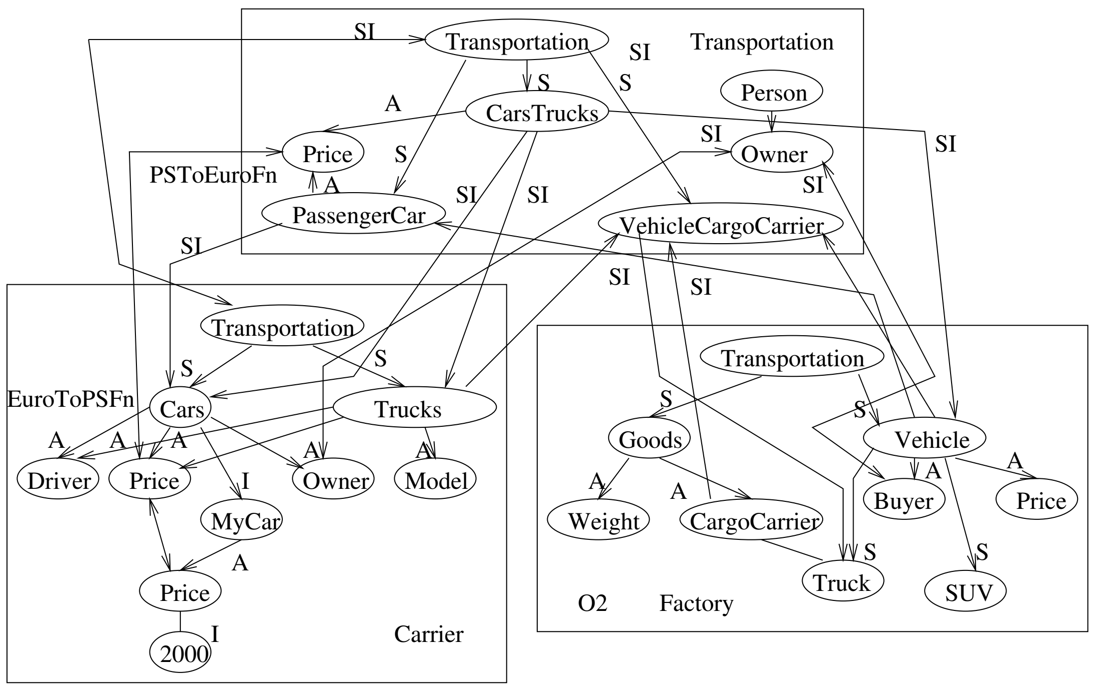


Figure 2: Articulation of Ontologies

### The Graph-Oriented Model

Formally, an ontology $O$ is represented by a directed labeled graph $G = (N, E)$ where $N$ is a finite set of labeled nodes and $E$ is a finite set of labeled edges. An edge $e$ is written as $(n_1, \alpha, n_2)$ where $n_1$ and $n_2$ are members of N and $\alpha$ is the label of the edge. The label of a node $n$ is given by a function $\lambda(n)$ maps the node to non-null string. In the context of ontologies, the label often maps to a noun-phrase. The label $\alpha$ of an edge $e$ = *(n1, $\alpha$, n2)* is a string given by $\alpha = \delta(e)$. The string attached to an edge relates to either a verb in natural language or a pre-defined semantic relationship. The domain of the functions $\lambda$ and $\delta$ is the universal set of all nodes (from all graphs) and the range is strings (from all lexicons).

*Example* Our running example already elicits the graphical structure envisioned. It is composition of three independent graphs, which each provide a ontological context. The semantic model for the sources is built around the relationships { "InstanceOf", "SubclassOf", "AttributeOf" }, which are commonly found in literature. The semantic bridges are indicated with dotted lines and will be discussed shortly.

**The Graph Patterns**

A necessary ingredient to manipulate the graphs is a mechanism to identify portions of interest in a concise manner. Graph patterns can be used for this purpose. We define a pattern $P$ to be a graph $P = (N', E')$, which matches a subgraph if, apart from a structural match, the labels of the corresponding nodes and the edges are identical. Formally, graph $G_1 = (N_1, E_1)$ is said to match into $G_2 = (N_2, E_2)$ if there exists a total *mapping function* $f : N_1 \rightarrow N_2$

1. for each node $n_1 \in N_1$, $\lambda_1(n_1) = \lambda_2(F(n_1))$.
2. for each edge $e_1 = (n_1, \alpha, n_2) \in E_1$, there exists an edge $e_2 = (f(n_1), \alpha, f(n_2)) \in E_2$.

In the ONION toolkit, the patterns are mostly identified by direct manipulation of the graph representation. For the textual interface we use a straight forward notation with (curly) brackets to denote hierarchical objects. Variables are indicated with bounded terms.

*Example* Possible patterns over our transportation world are $carrier.car.driver$, and $truck(O : owner, model)$ where variable $O$ binds with a truck owner object. Space limitations prohibit a further expose of the query facilities using patterns. We refer interested readers to papers on semi-structured query languages[17, 1].

**Graph Transformation Primitives**

In order to transform the ontology graphs we define four primitive operations: addition of nodes, deletion of nodes, addition of edges and deletion of edges. Addition of nodes and edges is used to generate the articulation and deletion is required to change it in response to changes in the underlying ontologies. The context of the operations is a graph $G = (M, E)$ where $M$ is the set of nodes $m_1, m_2, \cdots, m_n$ and subscripts $i, j, k \in \{1 \cdots n\}$. The operations are shortly defined as follows:

- *Node addition*

  Given the graph $G$, a node $N$ and its adjacent edges $\{(N, alpha_i, m_j)\}$ to add, the $NA$ operation results in a graph $G' = (M', E')$ where $M' = M \cup N$ and $E' = E \cup \{(N, alpha_i, m_j)\}$

- *Node deletion*

  Let $N \epsilon M$ be the node to be deleted and $Z = \{(N, \alpha_i, m_j\} \cup \{m_j, \alpha_i, N)\}$ the edges incident with $N$, then the node deletion operation $ND$, on the graph G, results in a graph $G' = (M', E')$ where $M' = M - N$ and $E' = E - Z$.

- *Edge Addition* Given a graph and a set of edges $SE = \{(m_i, \alpha_j, m_k)\}$ to add the edge addition operation $EA[G, SE]$ results in a graph $G' = (M, E')$ where $E' = E \cup SE$.

- *Edge Deletion* Given a graph and a set of edges $SE = \{(m_i, \alpha_j, m_k)\}$ to remove the edge deletion operation $ED[G, SE]$ results in a graph $G' = (M, E')$ where $E' = E - SE$.

For the sake of clarity, in the rest of the discussion, we will refer to edges using node labels instead of the nodes they represent, i.e., instead of saying edge $e = (n_1, \alpha, n_2)$ where $\lambda(n_1) = A$ and $\lambda(n_2) = B$ we will refer to it as the edge $e = (A, \alpha, B)$. This is, typically, not a problem for ontologies that are consistent, since, in the subclass hierarchies, a term is represented by one node.

## 4 Articulation of Ontologies

The ontologies in our running example represent three sections of the real world. The *carrier* and *factory* represent two autonomous knowledge sources, while the *transport* ontology models an articulation of the semantic interface necessary to relate the other sources. It does not stand on its own, but captures the semantic objects that help bridge the semantic gap between *carrier* and *factory*.

This observation has some far-reaching consequences. It reduces the articulation problem to identifying semantically relevant classes and to maintaining the subset-relationship with the underlying sources as semantic bridges. The focus of this section is to build mechanisms to carve out portions of an ontology, required by the articulation, using graph patterns. To complete the model, we introduce functional abstractions to convert information as required by the articulation process.

### 4.1 Ontology Terms and Patterns

An articulation graph $OA$ is built from structures taken from the underlying sources and maintains information regarding the relationships that exist between them. $OA$ is constructed using both interactive guidance from the domain expert and deployment of general articulation rules drawn from a knowledge base using SKAT. Such articulation rules take the form of $P \Rightarrow Q$ where $P, Q$ are complex graph patterns.

The construct $P \Rightarrow Q$ is read as "an $Q$ object semantically belongs to the class $P$", or " $P$ semantically implies $Q$". In a pure object-oriented setting, this amounts to restricting the semantic bridges considered to be a "directed subset" relationship. We shortly discuss the kind of articulation rules encountered and the method to represent them in ONION.

#### Semantic Implication Bridges

The rule $O_1.A \Rightarrow O_2.B$ where A and B are simple node identifiers is cast into the single edge (A, "SIBridge", B) between the ontology structures. It models the case that an $A$ object is a semantic specialization of $B$ and is the simpliest semantic bridge considered.

Prefixing the terms with their respective ontologies is a consequence of a linear syntax adhered to in this paper. In ONION a simple click and drag approach resolves this naming problem.

*Example.* The articulation rule ($carrier.Car \Rightarrow factory.Vehicle$) is translated to an edge addition operation, i.e. $EA[OU, \{(carrier.Car, "SIBridge", transport.Vehicle)(factory.Vehicle, "SIBridge", transport.Vehicle)(transport.Vehicle, "SIBridge", factory.Vehicle)\}]$. The first edge indicates that $carrier.Car$ is a specialization of $transport.Vehicle$. The other two edges establish the equivalence of $factory.Vehicle$ and $transport.Vehicle$

In spite of the term $Vehicle$ not occurring in $carrier$, modeling of such an articulation enables us to use information regarding cars in $carrier$ and to integrate knowledge about all vehicles from $carrier$ and $factory$ (at least as far as this is semantically valid).

*Example.* Alternatively, new terms can be added to the articulation graph using the cascaded short hand ($carrier.Car \Rightarrow transport.PassengerCar \Rightarrow factory.Vehicle$). To model this rule, the articulation generator adds a node PassengerCar to the $transport$ ontology. It then adds the edges $(carrier.Car, "SIBridge", transport.PassengerCar)$ and $(transport.PassengerCar, "SIBridge", factory.Vehicle)$.

Articulation rules are not confined to bridging ontologies. They can also be used to structure the articulation graph itself, in order to obtain a more coherent description. Likewise, the notational convenience of multi-term implication is broken down by articulation generator into multiple atomic implicative rules.

*Example.* The rule ($transport.Owner \Rightarrow$ transport.Person$) results in the addition of an edge, to the articulation ontology graph,$transport$, indicating that the class$Owner$is a subclass of the class$Person.

### Conjunction and disjunction

The operands for the semantic implication can be generalized to encompass graph pattern predicates. Their translation into the ONION data layer amounts to introducing a node to represent the sub-class derived and taking this as the target for the semantic implication. A few examples suffice to illustrate the approach taken.

*Example* The compound rule $((factory.CargoCarrier \wedge factory.Vehicle) \Rightarrow carrier.Trucks)$ is modeled by adding a node, $N$, to represents all vehicles that can carry cargo, to the articulation ontology. The default label for $N$ is the predicate text, which can be overruled by the user using a more concise and appropriate name for the semantic class involved. In our example, we introduce a node labeled $CargoCarrierVehicle$ and edges to indicate that this is a subclass of the classes $Vehicle$, $CargoCarrier$ and $Trucks$. Furthermore, all subclasses of $Vehicle$ that are also subclasses of $CargoCarrier$, e.g, $Truck$, are made subclasses of $CargoCarrierVehicle$. This is intuitive since a $CargoCarrierVehicle$ is indeed a vehicle, it carries cargo and is therefore also a goods vehicle.

Articulation rules that involve disjunction of terms, like, $(factory.Vehicle \Rightarrow (carrier.Cars \vee carrier.Trucks))$ are modeled by adding a new node in the articulation ontology labelled $CarsTrucks$ and edges that indicate that the classes $carrier.Cars$, $carrier.Trucks$ and $factory.Vehicle$ are subclasses of $transport.CarsTrucks$. Intuitively, we have introduced a term $CarsOrTrucks$ which is a class of vehicles that are either cars or trucks and the term $Vehicle$ implies (is a subclass of) $CarsOrTrucks$.

### Functional Rules

Different ontologies often contain terms for the same concept, but are expressed in a different metric space. Normalization functions can be used to associate such terms in a convenient way.

*Example.* The price of cars expressed in terms of Dutch Guilders and Pound Sterling might need to be normalized with respect to, say the Euro, before they can be integrated. The choice of the Euro - the normalized currency - is made by the expert (or politician!). We expect the expert to also supply the functions to perform the conversions both ways i.e. from Dutch Guilders to Euro and back.

Given the ontology graphs, and rules like $(DGToEuroFn() : carrier.DutchGuilders \Rightarrow transport.Euro)$, we create an edge $(carrier.DutchGuilders, "DGToEuroFn()", transport.Euro)$ to the articulation ontology from $carrier$. The query processor will utilize these normalizations functions to transform terms to and from the articulation ontology in order to answer queries involving the prices of vehicles.

### 4.2 Structure of the Articulation Ontology

The construction of the articulation ontology, as detailed above, mainly involved introducing nodes in the articulation ontology and edges between these nodes and nodes in the source ontologies. There are very few edges between the nodes in the articulation ontology, unless explicit articulation rules were supplied linking them. Such implication rules are essential if the articulation expert envisages a new structure for the articulation ontology. The expert can cut and paste portions of $O_i$ and indicate that the structure of OA is to be similar to these portions using either the graphical interface or pattern-based rules. We then generate the structure between the nodes in the articulation ontology based primarily on the transitive closure of the edges in the ontology O1 and other inference among the articulation rules although all transitive semantic implications are not shown unless requested by the expert.

## 5 An Ontology Algebra

We define an algebra to enable interoperation between ontologies using the articulation ontology, The input to the *operators* in the algebra are the ontology graphs. *Unary* operators like *filter* and *extract* work on a single ontology. They are analogous to the *select* and *project* operations in relational algebra. They help us define the interesting areas of the ontology that we want to further explore. Given an ontology and a graph pattern an unary operation matches the pattern and returns selected portions of the ontology graph. *Binary* operators include *union*, *intersection* and *difference*. They take as input two ontologies and the articulation rules and output another ontology constructed based on the articulation.

### 5.1 Union

Answering user queries involves consulting more than one knowledge source and composing knowledge from them if required. In the most general case, where the query-plan indicates that more than one knowledge base needs to be consulted, queries should be directed to the union of the ontologies. We use the two sources and the articulation ontology to glue the sources together.

The union operator takes two ontology graphs, a set of articulation rules and generates a unified ontology graph where the resulting unified ontology comprises of the two original ontology graphs connected by the ontology articulation as outlined in the previous section.

The ontology union $OU$ of source ontologies $O1$ and $O2$ is defined as $O1 \cup_{rules} O2 = OU$. Let $O1 = (N1, E1), O2 = (N2, E2)$ be the graphs representing the source ontologies, $OA = (NA, EA)$ represents the articulation ontology, BridgeEdges is the set of edges connecting nodes between $OA$ and either $O1$ or $O2$, as computed by the articulation generator to model the articulation rules and $OU$ be the graph representing the unified ontology. $OU = (N, E, )$ is such that $N = N1 \cup N2 \cup NA$ and $E = E1 \cup E2 \cup EA \cup BridgeEdges$

### 5.2 Intersection

The intersection operator takes two ontology graphs, a set of articulation rules and produces the articulation ontology graph. The articulation ontology graph consists of the nodes added by the articulation generator representing the articulation rules and the edges between these nodes. The edges that are between these nodes and nodes in the source ontologies are not included since those nodes are not part of the articulation ontology graph. The intersection, therefore, produces an ontology that can be further composed with other ontologies.

The ontology intersection $OI$ of source ontologies $O1$ and $O2$ is defined as $O1 \cap_{rules} O2 = OI$ where $OI = OA$ as generated by the articulation generator.

### 5.3 Difference

The difference of two ontologies $(O1 - O2)$ is defined as the terms and relationships of the first ontology that have not been determined to exist in the second.

*Example.* Assume the only articulation rule that exists is $(carrier.Car => factory.Vehicle)$. It is intuitively clear that the difference between the ontologies $carrier$ and $factory$ should not contain cars. Therefore, the articulation generator deletes the node $Car$ and all nodes that can be reached by a path from $Car$, but not by a path from any other node. All edges incident with any deleted node is also deleted.

Now the difference $(factory - carrier)$ will contain the node 'Vehicle' (provided no other rule indicates that an equivalent class or superclass of vehicle exists in $carrier$.) Although, the the first source contains knowledge about cars, which are vehicles, the expert rule does not identify which vehicles in the second source are cars. To compute the difference, only cars need to be deleted from the second source and not any other type of vehicle. Since, with the given rules, there is no way to distinguish the cars from the other vehicles in the second knowledge source, the articulation generator takes the more conservative option of retaining all vehicles in the second ontology. Therefore, the node 'Vehicle' is not deleted.

The difference, OD, of two ontologies O1 and O2 is defined as O1 - O2 = OD. Let O1=(N1, E1), O2=(N2, E2) be the graphs representing the source ontologies, OA=(NA, EA) is the graph representing the articulation ontology. OD=(N,E) represents the ontology OD such that n belongs to N if

- n belongs to N1 and n does not belong to N2

- and $\forall n'$ belonging to NA, no rule of the form $(O1.n => OA.n')$ exists.

and an edge $e \epsilon E$ if $e \epsilon E1$ and if $e = (n1, \alpha, n2)$, $n1 \epsilon N$ and $n2 \epsilon N$.

## 6 Conclusion

We have outlined a scalable framework for a system that enables interoperation between knowledge sources to reliably answer user queries. The approach ensures minimal coupling between the sources, so that the sources can be developed and maintained independently. We believe that this approach greatly reduces costs to the applications composing knowledge from a large number of sources that are frequently updated like the world-wide web.

This paper highlights a sound formalism used to represent ontologies graphically. Resolution of semantic heterogeneity is addressed using expert rules. Semantic relationships are first represented using first-order logic based articulation rules. These rules are then modeled using the same graphical representation. This representation is simple, easy to visualize and provides the basis for a tool that generates the articulation by resolving semantic heterogeneity semi-automatically.

The main innovation in ONION is that it captures semantic relationships using logical rules concisely. We then show how this can be modeled using a graph-oriented model as the data layer. This clean representation helps in separating the data layer with the inference engine.

The system architecture provides the ability to plug in different semantic reasoning components and inference engines which can make the task of the expert easier. How such components can use external knowledge sources and lexicons to suggest a better articulation is being currently investigated as part of completing the implementation of the ONION toolkit.

## 7 Acknowledgements

Thanks are due to Jan Jannink for his help in developing some of the basic ideas.

## References

[1] Extensible markup language (xml) 1.0 http://www.w3.org/tr/rec-xml, Feb 1999.

[2] C.A. Knoblock, S. Minton, J.L. Ambite, N. Ashish, P.J. Modi, Ion Muslea, A.G. Philpot, and S. Tejada. Modeling web sources for information integration. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence, Madison, WI*, 1998.

[3] P. Buneman, S. Davidson, and A. Kosky. Theoretical aspects of schema merging. In G. Goos and J. Hartmanis, editors, *Advances in Database Technology - EDBT '92, Proc. 3rd International Conference on Extending Database Technology*, number 580 in Lecture Notes in Computer Science, pages 152–167. EDBT, Springer-Verlag, Mar. 1992.

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

[5] M.R. Genesereth, A.M. Keller, and O.M. Duschka. Infomaster: An information integration system. In *ACM SIGMOD '97*, pages 539–542. ACM, 1997.

[6] The information manifold, http://portal.research.bell-labs.com/orgs/ssr/people/levy/paper-abstracts.html#iga.

[7] A.T. McCray, A.M. Razi, A.K. Bangalore, A.C. Browne, and P.Z. Stavri. The umls knowledge source server: A versatile internet-based research tool. In *Proc. AMIA Fall Symp*, pages 164–168, 1996.

[8] The stanford-ibm manager of multiple information sources, http://www-db.stanford.edu/tsimmis/.

[9] P. Scheuermann, C. Yu, A. Elmagarmid, H. Garcia-Molina, F. Manola, D. McLeod, A. Rosenthal, and M. Templeton. Report on the workshop on heterogenous database systems. In *ACM SIGMOD RECORD 19,4*, pages 23–31, December 1989.

[10] A.P. Sheth and J.A. Larson. Federated database systems for managing distributed, heterogenous, and autonomous databases. In *ACM Computing Surveys*, pages 183–236, 1994.

[11] M. Siegel and S. Madnick. A metadata approach to solving semantic conflicts. In *Proc. of the 17th International Conference on Very Large Data Bases*, pages 133–145, 1991.

[12] The context interchange project, http://context.mit.edu/ coin/.

[13] Cyc knowledge base, http://www.cyc.com/.

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

[15] Ontolingua, http://www-ksl-svc.stanford.edu:5915/doc/project-papers.html.

[16] P Mitra, G Wiederhold, and J Jannink. Semi-automatic integration of knowledge sources. In *Proc. of the 2nd International Conference On Information Fusion - FUSION'99*, July 1999.

[17] Y. Papakonstantinou, H. Garcia-Molina, and Widom J. Object exchange across heterogeneous information sources, March 1995.