A Graph-Oriented Model for Articulation of Ontology Interdependencies*

Prasenjit Mitra[1], Gio Wiederhold[1], and Martin Kersten[3]

[1] Stanford University, Stanford, CA, 94305, U.S.A.
[2] INS, CWI, Kruislaan 413, 109 GB Amsterdam, The Netherlands
mitra,gio@db.stanford.edu mk@cwi.nl

Abstract. *Ontologies* explicate the contents, essential properties, and relationships between terms in a knowledge base. Many sources are now accessible with associated ontologies. Most prior work on the 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 domain 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 between heterogenous sources to answer end-user queries 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 the *relationships* among them. Ontologies relate to knowledge sources like dictionaries relate to literary works. Like the dictionary, the ontology collects and organizes the terms

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

of reference. By analogy, definitions in a dictionary give us the relationships between words while ontologies give us the relationships between terms.

The ontologies considered in this paper 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 achieved. Today, XML [1] is used as a carrier of semantic information, but by itself an XML representation is not sufficient. An XML document can only represent a single domain ontology but can do little to resolve errors introduced by semantic mismatches. We focus on such structured worlds as a starting point, establish how to build *semantic bridges*, i.e, logical rules that bridge the semantic gap between sources and use reasoning based on this semantic information to compose knowledge from multiple knowledge sources.

Bridging the gap between multiple information sources is 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 database schema. However, unification of schemas does not scale well since broad schema integration leads to huge and difficult-to-maintain schemas. Since the meaning and scope of the database concepts is not made explicit and is likely to differ it very quickly, calls for identification of 'sub-domains' and 'namespaces' [4] to help 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 can remain independent. In most real-life situations, it suffices to just use semantic bridges wherever interaction among information sources is required.

Recent progress in automated support for mediated systems, using 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. Defining such views, however, requires manual specification. Views need to be updated or reconstructed even for small changes to the individual sources. Instead of manually creating and materializing such views, we provide a semi-automatic rule-based framework for interoperation based on articulations of ontologies. Our contention is that, in many cases, an 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 heterogenous databases [9],[10] and multidatabase systems [11], [12]. One of the strategies used is to merge all system schemas into a global reference schema. Such a strategy suffers the same drawbacks as the information integration approach. Furthermore, relying on the end-user to bridge the semantic gap between information obtained from multiple databases imposes the implicit assumption that all end-users are domain experts. Instead, we envision a system to propose and resolve the semantic gaps only in the intersection among the knowledge sources that is relevant to the application. Such a system is driven by a rule set supplied by the domain interoperation expert, who focuses on creating an articulation. The tedious task of creating an articulation 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],[14]. While a text-based model is easy to construct, its structural relationships is often hard to visualize. 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 [15] 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 novelty 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. Its modular framework allows for a clean separation of the several knowledge processing components. The model is simple, yet rich enough, to provide a basis for the logical inference necessary for knowledge composition and for the detection of errors in the articulation rules. An ontology algebra is defined, which is the machinery to support the composition of ontologies via the articulation. The implementation of the ONION system is based on the ontology algebra.

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

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 Fig. 1, the unified ontology $Ont5$ contains to the source ontologies $Ont3$ and $Ont4$ and the articulation ontology $Art2$. The articulation of the source ontologies $Ont4$ and $Ont3$ consists of the articulation ontology $Art2$ and the semantic bridges linking it to the source ontologies.

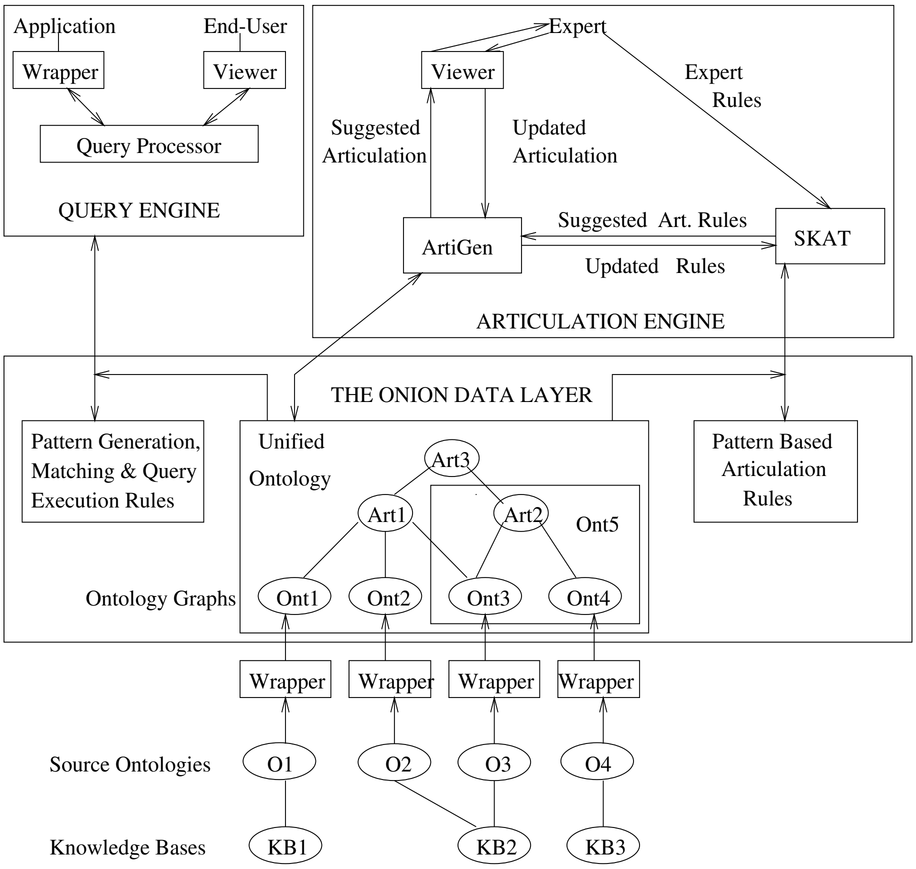

Fig. 1. The ONION system

We now present an overview of the system architecture and introduce a running example. The ONION architecture, shown in Fig. 1, has been designed with strong modularity in mind. 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, represented as a graph, 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 [14]. Our approach is to separate the logical inference engine from the representation model for the ontologies as much as possible. This allows us to accommodate different inferences engines that can reason about different ontology graphs and ensures that we do not mandate a particular semantics for logical reasoning.

An articulation is also represented as an ontology graph along with structures A_ij, i.e., the semantic bridges that link the articulation ontology to its underlying ontologies O_i and O_j. The articulation ontology commonly uses concepts and structures inherited from the individual 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, most importantly, specify articulation rules.

The alternative method is to call upon the articulation generator 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.

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 [16] and generate its solution.

2.4 The Articulation Engine

The articulation engine is responsible for creating the articulation ontology and the semantic bridges betwen it and the the source ontologies based on the articulation rules. ONION is based on the SKAT (Semantic Knowledge Articulation Tool) system developed in recent years at Stanford [17]. 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. The inference engine uses the articulation rules generated by SKAT and the rules from the individual source ontologies to derive more rules if possible.

The articulation generator takes the articulation rules and generates the articulation, i.e., the articulation ontology graph and the semantic bridges, which is then forwarded to the expert for confirmation. The expert has the final word on the articulation generation and is responsible to correct inconsistencies in the suggested articulation. If the expert suggests modifications 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.

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) (Fig. 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 the sake of clarity a few of the most obvious edges have been omitted. Apart from the above mentioned relationships, the individual ontologies also contain other binary relationships between terms. The ontologies are expected to have rules that define the properties of each realtionship, e.g., we will have rules that indicate the transitive nature of the 'SubclassOf' relationship. These rules are used by the articulation generator and the inference engine while generating the articulation and also while answering end-user queries.

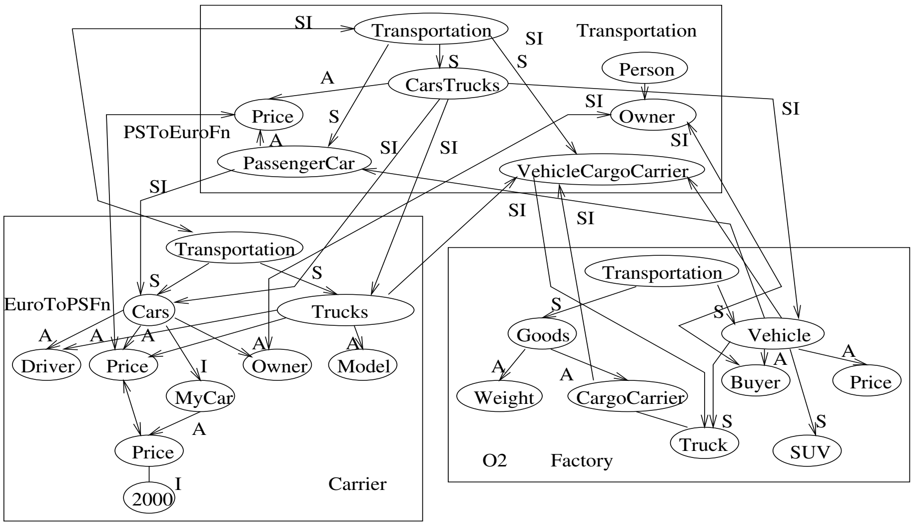

Fig. 2. Articulation of Ontologies

3 Graphical Representation of Ontologies

The ONION system has been anchored in the seminal work on graph-based databases in [15]. In this section we introduce its formal setting and the associated graph operations.

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, α, n_2) where n_1 and n_2 are members of N and α is the label of the edge. The label of a node n is given by a function $\lambda(n)$ that maps the node to non-null string. In the context of ontologies, the label often maps to a noun-phrase that represents a concept. The label α 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 λ and δ is the universal set of all nodes (from all graphs) and the range is the set of strings (from all lexicons).
Example Our running example already elicits the graphical structure envisioned. It is a composition of three independent graphs, each of which provides an ontological context. The semantic model for the sources is built around the relationships { "InstanceOf", "SubclassOf", "AttributeOf" }, which are commonly found in literature.

The Graph Patterns. To manipulate the ontology graphs we need to identify portions of the graphs that are 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* $f : N_1 \rightarrow N_2$ such that

1. $\forall n_1 \in N_1, \lambda_1(n_1) = \lambda_2(f(n_1))$.
2. $\forall e_1 = (n_1, \alpha, n_2) \in E_1, \exists e_2 = (f(n_1), \alpha, f(n_2)) \in E_2$.

In practice, apart from the strict match described above, the domain interoperation expert can define versions of fuzzy matching. For example, the expert can indicate a set of synonyms and provide a rule that would relax the first condition and enable nodes to match not only if they have the exact same label but also if they are synonyms as defined by the expert. Alternatively, the second condition that requires edges to have the same label may not be strictly enforced.

In the ONION toolkit, the patterns are mostly identified by direct manipulation of the graph representation. For the textual interface we use a simple 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*$(O : owner, model)$. The former partially specifies a pattern in the *carrier* ontology that consists of a node *car* which has an outgoing edge to the node *driver*. The latter refers to a a node labelled *truck* that has attributes *owner* and *model* and the variable O binds with the 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 [18, 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 while generating the articulation. Deletion is required while updating the articulation 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 use a node's label in place of the node, while referring to an edge from(or to) that node, 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 consistent ontologies where a term (representing a concept) is depicted by one node in the ontology graph. Thus we use the term(label) interchangeably with the node related to it.

4 Articulation of Ontologies

The ontologies in our running example represent three sections of the real world. The *carrier* and *factory* ontologies represent two autonomous knowledge sources, while the *transport* ontology models an articulation of the individual source ontologies and provides the necessary semantic interface to relate the 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 two tasks that are performed by the articulation generator with advice from the domain expert who is knowledgeable about the semantics of the two ontologies being articulated. The first task is to identify semantically relevant classes to include in the articulation ontology. The second task is to generate and maintain the semantic bridges, i.e., the subset-relationships between articulation classes and the related classes in the underlying source ontologies. Since the generation of the articulation is semi-automatic, the developers and users of the ontologies do not have to keep track of the semantic differences between the different ontologies. 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 "the object Q 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
$(transport.PassengerCar, "SIBridge", factory.Vehicle)$.

Rules are not confined to describing bridging ontologies but are also used to structure the individual source ontologies or the articulation ontology graph itself. Likewise, the notational convenience of multi-term implication is broken down by the inference engine 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 subclass 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, to 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) $CarsOrTrucks$.

Since inference engines for full first-order systems tend not to scale up to large knowledge bases, for performance reasons, we envisage that for a lot of applications, we will use simple Horn Clauses to represent articulation rules. The modular design of the ONION system implies that we can then plug in a much lighter (and faster) inference engine.

Functional Rules

Different ontologies often contain terms that represent the same concept, but are expressed in a different metric space. Normalization functions, that take in a set of input paramters and perform the desired conversion are written in a standard programming language and provided by the expert. There is scope for the generation of such functions semi-automatically in the future.
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 to 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 to create the edges. Such implication rules are especially essential if the articulation expert envisages a new structure for the articulation ontology. The expert can select portions of O_i and indicate that the structure of OA is similar to these portions using either the graphical interface or pattern-based rules. The articulation generator, then generates the edges between the nodes in the articulation ontology based primarily on the edges in the selected portion of O_i, the transitive closure of the edges in it and other inference using the articulation rules although all transitive semantic implications are not displayed by the viewer unless requested by the expert.

It is important to note that the articulation ontology of two ontologies can be composed with another source ontology to create a second articulation that spans over all three source ontologies. This implies that with the addition of new sources, we do not need to restructure existing ontologies or articulations but can reuse them and create a new articulation with minimal effort.

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 [19]. 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*. Each operation is defined on two ontologies (and the articulation rules) and results in an ontology that can be further composed with other ontologies.

5.1 Union

If the query-plan generated while answering user queries indicates that more than one knowledge base needs to be consulted, queries are directed to the union of the ontologies. 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 articulation. The articulation is generated using the articulation rules as outlined in the previous section.

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

The union of the $carrier$ and $factory$ ontologies is the $carrier$ and $factory$ ontologies themselves along with the articulation ontology $transportation$ and the edges connecting the $transportation$ ontology to the source ontologies.

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 using the articulation rules and the edges between these nodes. In order to ensure that the edges in the articulation ontology connect nodes that are in the articulation ontology graph, the edges that are between nodes in the articulation ontology graph and nodes in the source ontology graphs are not included in the articulation ontology if the nodes in the source ontology graphs are not part of the articulation ontology graph. The intersection, therefore, produces an ontology that can be further composed with other ontologies. This operation is central to our scalable articulation concepts.

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.

The intersection of the *carrier* and $factory$ ontologies is the $transportation$ ontology.

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. This operation allows a local ontology maintainer to determine the extent of one's ontology that remains independent of the articulation with other domain ontologies. If a change to a source ontology, say $O1$, occurs in the difference of $O1$ with other ontologies, no change needs to occur in any of the articulation ontologies. If on the other hand a node occurs in $O1$ but not in $O1 - O2$ then any change related to the node or any edges connecting it to other nodes must also be reflected in the articulation ontologies.

Example. Assume the only articulation rule that exists is (*carrier.Car* $=>$ $factory.Vehicle$), i.e, a Car is a $Vehicle$. It is intuitively clear that to obtain the difference between the ontologies *carrier* and $factory$, we should subtract all vehicles from $carrier$. Since a Car is a $Vehicle$, *carrier* should not contain Car. Therefore, to obtain the difference between *carrier* and $factory$, the articulation generator deletes the node Car from *carrier* 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. Like the Union, the Difference is computed dynamically and is not physically stored.

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 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.

Let $O1 = (N1, E1), O2 = (N2, E2)$ be the graphs representing the source ontologies, $OA = (NA, EA)$ is the graph representing the articulation ontology. The difference, $OD = O1 - O2$ is a represented by a graph (N, E) such that $n \in N$ only if

1. $n \in N1$ and $n \notin N2$.
2. and there exists no path from n to any n' in $N2$.

and $e \in E$ only if

1. $e \in E1$
2. and if $e = (n1, \alpha, n2)$, $n1 \in N$ and $n2 \in N$.

The algebra forms the basis for the ONION system. The intersection determines the portions of knowledge bases that deal with similar concepts. The union of knowledge bases presents a coherent, connected and semantically sound unified knowledge base that is computed dynamically in response to queries. The difference provides us the portions of the knowledge bases that can be independently manipulated without having to update any articulation.

6 Conclusion

We have outlined a scalable framework for a system that enables interoperation between knowledge sources to reliably answer user queries. The main innovation in ONION is that it uses articulations of ontologies to interoperate among ontologies. A semi-automatic approach ensures that most of the work involved in interoperation is automated, yet an easy-to-use graphical interface is provided to assist the expert in making sure that the system is reliable.

The approach ensures minimal coupling between the sources, so that the sources can be developed and maintained independently. Changes to portions of an ontology that are not articulated with portions of another ontology can be made without effecting the rest of the system. This approach greatly reduces the cost of maintenaning applications that compose knowledge from a large number of sources that are frequently updated like those in the world-wide web.

This paper highlights a formalism used to represent ontologies graphically. This clean representation helps in separating the data layer with the inference engine. 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 semi-automatically.

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 *Proc. of the Fifteenth National Conf. on Artificial Intelligence, Madison, WI*, 1998.
[3] P. Buneman, S. Davidson, and A. Kosky. Theoretical aspects of schema merging. In *Proc. of EDBT '92*, 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] Information integration using infomaster, http://infomaster.stanford.edu/infomaster-info.html.
[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, 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 Int. Conf. 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] Ontolingua, http://www-ksl-svc.stanford.edu:5915/doc/project-papers.html.
[15] M. Gyssens, J. Paredaens, and D. Van Gucht. A graph-oriented object database model. In *Proc. PODS*, pages 417–424, 1990.
[16] P. Mitra. Algorithms for answering queries efficiently using views, http://www-db.stanford.edu/ prasen9/qralgo.pdf. Technical report, Infolab, Stanford University, September 1999.
[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] Y. Papakonstantinou, H. Garcia-Molina, and Widom J. Object exchange across heterogeneous information sources, March 1995.
[19] J. Jannink. *Resolution of Semantic Differences in Ontologies.* PhD thesis, Stanford University.