Journal of Intelligent Information Systems, 16, 149–187, 2001

Compiling Source Descriptions for Efficient and Flexible Information Integration

JOSÉ LUIS AMBITE ambite@isi.edu
CRAIG A. KNOBLOCK knoblock@isi.edu
ION MUSLEA muslea@isi.edu
ANDREW G. PHILPOT philpot@isi.edu
Information Sciences Institute, Integrated Media Systems Center, and Department of Computer Science, University of Southern California, 4676 Admiralty Way, Marina del Rey, CA 90292

Abstract. Integrating data from heterogeneous data sources is a critical problem that has received a great deal of attention in recent years. There are two competing approaches to address this problem. The traditional approach, which first appeared in Multibase and more recently in HERMES and TSIMMIS, often called global-as-view, defines the global model as a view on the sources. A more recent approach, sometimes referred to as local-as-view or view rewriting, defines the sources as views on the global model. The disadvantage of the first approach is that a person must re-engineer the definitions of the global model whenever any of the sources change or when new sources are added. The view rewriting approach does not suffer from this drawback, but the problem of rewriting queries into equivalent plans using views is computationally hard and must be performed for each query at run-time.

In this paper we propose a hybrid approach that amortizes the cost of query processing over all queries by pre-compiling the source descriptions into a *minimal* set of integration axioms. Using this approach, the sources are defined in terms of the global model and then compiled into axioms that define the global model in terms of the sources. These axioms can be efficiently instantiated at run-time to determine the most appropriate rewriting to answer a query and facilitate traditional cost-based query optimization. Our approach combines the *flexibility* of the local-as-view approach with the run-time *efficiency* of the query processing in global-as-view systems. We have implemented this approach for the SIMS and Ariadne information mediators and provide empirical results that demonstrate that in practice the approach scales to large numbers of sources and that the approach can compile the axioms for a variety of real-world domains in a matter of seconds.

Keywords: information integration, information mediators, axiom compilation, heterogeneous data sources, SIMS, Ariadne

1. Introduction

The problem of integrating data from heterogeneous collections of sources is ubiquitous. With the rise of the Internet as well as intranets the number of available and relevant sources to an organization continues to grow. One effective solution to this problem is the development of information mediators (Wiederhold, 1992). An information mediator provides seamless access to a collection of related, but possibly heterogeneous and distributed data sources. There are a variety of approaches to building information mediators, illustrated by different approaches used in systems such as TSIMMIS (Hammer et al., 1995), Garlic (Haas et al., 1997; Roth and Schwarz, 1997), HERMES (Adali et al., 1996), Information Manifold (Levy et al., 1995b), InfoSleuth (Bayardo et al., 1997), Infomaster (Duschka and

Genesereth, 1997), SIMS (Arens et al., 1993; Arens et al., 1996), and Ariadne (Knoblock et al., 1998; Knoblock et al., 2000) to name a few. However, one issue that is common to all of these approaches is how to scale these systems to large numbers of information sources in a way that is both computationally tractable and natural to the developers of new applications.

In information mediators, a central problem is how to efficiently process queries. This query optimization problem consists of both selecting a set of sources that can be used to answer a query and generating a cost-effective query access plan that specifies the order of retrieval and manipulations on the data. In this paper we focus on the first problem (which has been referred to as both source selection and query planning in the literature) for a mediator with an expressive language for describing the contents of sources. In a separate paper (Ambite and Knoblock, 2000) we present our approach to generating query plans using a cost-based optimizer that takes advantage of the source selection techniques presented here. In this paper we focus on the problem of how to compactly represent and efficiently use the alternative combinations of sources that can be used to answer queries posed to a mediator.

Our approach to source selection is to build a global domain model (sometimes referred to as a world model) for an application and describe the contents of each of the sources in terms of the domain model. Based on this model, our system compiles the definitions of each of the sources into a minimal set of axioms that describe the possible ways the sources can be combined to produce any of the information that may be requested for each class in the domain model. The compilation algorithm is incremental, which means that when new sources are added, the system can efficiently update the axioms. Our approach provides flexibility and source independence by describing sources in terms of the domain model and it provides an efficient mechanism to incorporate source selection into the query processing.

The solution presented in this paper can be viewed as a combination of two existing approaches:

Sources modeled as views on the domain model: this approach, also known as local-as-view or view rewriting (Levy et al., 1995b), has the advantage that each source is modeled independently of all the other sources, so new sources can be added and existing sources can be modified without changing the domain model. The disadvantage is that performing the view rewritings is computationally hard (Levy et al., 1995a) and finding a complete answer to a query requires computing query containment in both directions. Moreover, this expensive computation would have to be done repeatedly at query planning time. In contrast, our approach has the advantage of source independence without the computationally difficult problem of testing query containment during query planning.

The domain classes modeled as views on the source models: this approach, called global-as-view, (Landers and Rosenberg, 1982; Hammer et al., 1995; Roth et al., 1996) has the advantage that the sources required to provide the data for a specific class of information can be determined by simply looking up the definition of the domain class. The disadvantage is that it is difficult to construct and maintain the required axioms. For example, adding a new source to the system may require changes in many of these definitions.

Moreover, in previous systems these changes are done manually, which represents a time-consuming and error-prone process. In contrast, our approach provides the advantage of being able to quickly determine the combination of sources for a domain class since they are pre-compiled from the source definitions and avoids the disadvantage of having to manually build and maintain the definitions.

A specific instantiation of this general approach to source selection has been developed for the SIMS (Arens et al., 1993; Arens et al., 1996) and Ariadne (Knoblock et al., 1998) information mediators. This paper describes the language used in SIMS and Ariadne for defining sources, the algorithms for compiling the domain axioms from the source definitions, the approach to instantiating these domain axioms at run-time, and the relationships with previous work. We provide experimental results from a number of real-world applications to show that the axiom compilation runs in a matter of seconds in practice. We also provide experimental results on a set of synthetic domains to show that the axiom compilation algorithm scales to large domains that involve fifty or more interrelated sources, which match or exceed the complexity of real-world domains. Overall, our approach provides a simple, efficient, and elegant solution for integrating heterogeneous data sources.

2. Background

In the SIMS project we are addressing the problem of providing integrated access to heterogeneous distributed information sources. To build an application in SIMS, a user creates a *domain model* using the Loom (MacGregor, 1990) knowledge representation language and describes the source contents in terms of this model. The domain model establishes a fixed vocabulary describing object classes, their attributes, and the relationships among them. SIMS accepts queries in this domain-level language, processes these queries, and returns the requested data. Thus, the queries to SIMS do not contain information describing which sources are relevant to finding their answers or where they are located. Queries do not need to state how information obtained from different sources should be joined or otherwise combined or manipulated. It is the task of the system to determine how to efficiently and transparently retrieve and integrate the data necessary to answer a query.

In the previous work on SIMS (Arens et al., 1996) the selection of the sources was performed *dynamically* by searching the space of query reformulations given the domain model and source descriptions. This approach provided the flexibility we wanted in terms of dynamically selecting sources for answering queries; however, it did not scale well to large numbers of sources since the search space becomes quite large as the number of sources increases. The work described in the remainder of this paper extends our previous work by pre-compiling the source definitions into a set of domain axioms. The domain axioms compactly express the possible ways of obtaining the data for each class in the domain model. This approach provides the same flexibility and extensibility of the previous one and can perform the source selection much more efficiently.

In the remainder of this section, we review the language for describing an application domain, describe how the sources are defined in terms of the domain model, and then present a detailed motivating example that is used throughout the paper.

2.1. *Modeling the domain and the sources*

An information mediator typically has a representation of its domain of expertise, called the *domain model*, and descriptions of information sources in terms of the domain model. In SIMS, the domain model is specified in a subset of Loom (MacGregor, 1990), which is a KL-ONE style knowledge representation language (Brachman and Schmolze, 1985). KL-ONE style languages, also known as description logics, contain unary relations (*classes*), which represent the classes of the objects in the domain, and binary relations (*attributes*), which describe relationships between objects. Classes are defined using a set of class constructors. In the SIMS language, we support the following ways of constructing classes:

- **Primitive:** A primitive class is defined as a subclass without specifying the constraints that differentiate it from the parent class.
- **Defined:** A class can be defined using the following class constructors:
  - **Attribute introduction:** A class C can be defined as having an attribute R relating class C to another class D.
  - **Conjunction:** A class can be specified as a conjunction of other classes or constraints.
  - **Disjunction (Covering):** A class can be specified as a disjunction of its subclasses.
  - **Equality and Order Constraints:** A class can be specialized by conjoining it with order constraints of the type: attribute θ constant, where $\theta \in \{=, <, \leq, >, \geq\}$.

A set of attributes is associated with each class, and any subclass of a given class, C, inherits all of the attributes of C. For purposes of integration, we also require that every class has at least one defined key, which represents one or more attributes that uniquely identify the objects in a class. Since there may be more than one way to uniquely identify a object, a class may also have multiple defined keys.

In our approach, the domain model is used to describe the available information sources. The source description for source S with attributes $S.a_1, \ldots, S.a_n$ is written:

$$S(S.a_1, \ldots, S.a_n) \equiv D_S(a_1, \ldots, a_n).$$

The equation above specifies that the source S provides all instances of the domain class D_S with the corresponding attributes.[1] In contrast to other approaches to information integration (e.g., Information Manifold (Levy et al., 1995b)), which use *containment* to express the relationship between a class and its information sources, we assume that the source description defines *exactly* the class of information provided by the sources. This can be done without loss of generality because containment can be expressed by defining a subclass in the domain model. The use of exact descriptions has two major advantages: it supports complete answers to queries, and, when complete answers are not possible, it allows the system to determine when and in what way the answer is incomplete. A limitation in our approach is that one cannot describe a source as a join over domain classes. The reasons for this limitation and the applicability of our results are discussed in more detail in Section 7.

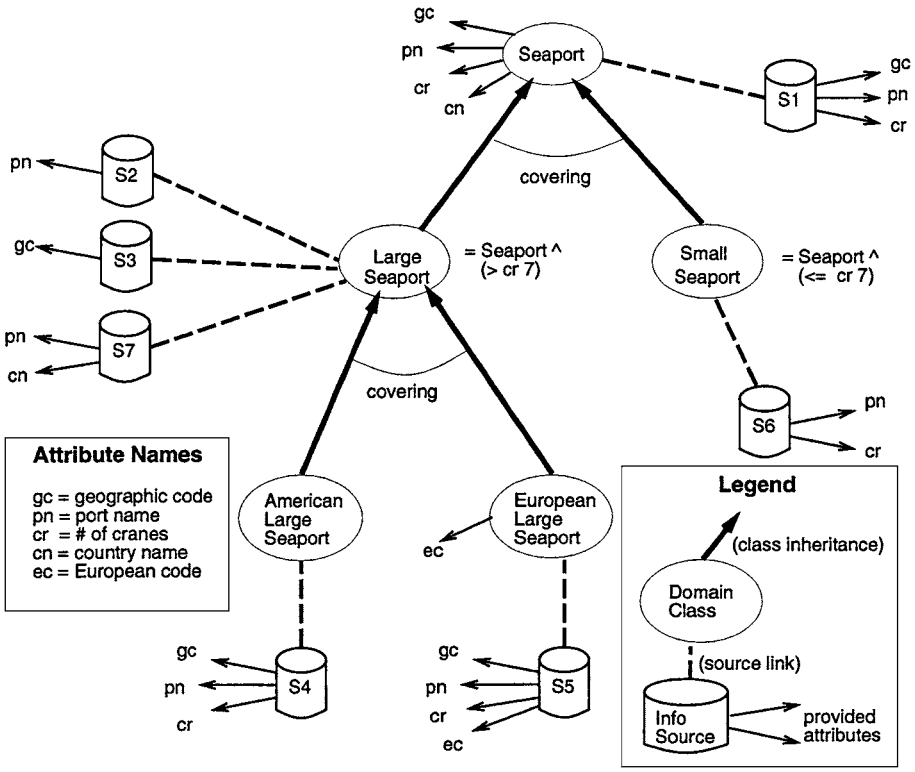

Figure 1. Example domain model and sources.

2.2. *Motivating example*

Consider a very simple application domain that contains a variety of information sources about different types of seaports. Figure 1 depicts the domain model constructed for this application domain. The domain classes are shown by ovals and are linked in an inheritance hierarchy. In the diagram, inheritance links are specified with solid arrows. In the example above we have two coverings: seaport is the union of large-seaport and small-seaport; and large-seaport is the union of american-large-seaport and european-large-seaport. The two subclasses in the first covering are created by specifying their exact relationship with their superclass (i.e., the seaports with at most/more than seven cranes), while american-large-seaport and european-large-seaport are introduced as primitive classes.

Every class has a corresponding set of attributes, shown by arrows, and classes also inherit all of the attributes of their ancestors. In our example, seaport, large-seaport, and small-seaport have four attributes: geographic location code (gc), port name (pn), country name (cn), and number of cranes (cr). Besides the inherited attributes, the class of european-large-seaport has an additional attribute, European code (ec), which is specific only to this class.

In addition to the domain model, the available sources are also shown in figure 1. They are represented by the database symbols linked to the corresponding domain classes by dashed lines. The link between the source and the domain class means that there is a one-to-one mapping between the instances of the domain class and the instances in the source. In our example, there are three sources for the class large-seaport: s2 provides the attribute pn, s3 provides gc, and s7 provides both pn and cn. For the sake of simplicity, each information source described here will be assumed to contain a single table; both the source and the table will be referred to by the same name. Since sources might provide only a subset of the possible attributes of a class, the specific attributes for each source are also shown in the figure.

There are a total of seven data sources, which provide different sets of attributes about the five domain classes (note that no source provides all attributes for a class). In the next section, we explain how the source definitions are compiled into a concise set of axioms that specify the ways in which individual sources can be combined so that they provide the data for all domain classes.

3. Compiling domain axioms from source definitions

A fundamental task of a mediator is to translate a query expressed in terms of the domain model into queries to the underlying sources. This involves finding the relevant combinations of sources that provide the attributes required by each class in a query. Instead of repeatedly searching at run-time for the combinations of sources that are relevant for each query, our system compiles in advance a set of *domain axioms* that compactly captures these combinations. By pre-compiling the axioms, the efficiency of query planning is improved dramatically because the mediator can use the readily available axioms as macro expansions instead of searching the space of all possible combinations of sources for the domain classes. Moreover, the compilation effort is amortized over all subsequent queries on the application domain. This section describes the details of how the domain axioms are pre-compiled and stored for efficient use at run-time.

A domain axiom specifies a particular way in which the available sources can be combined to provide the data for a domain class. For example, the port-name for the class large-seaport can be directly retrieved from the source s2, as can be seen in figure 1. This would be expressed as the axiom:

$$\text{large-seaport(pn)} \equiv \text{s2(s2.pn)}$$

There may be several axioms for a given class and set of attributes. For example, an alternative way of obtaining the same port names is to perform a union over sources s4 and s5, which provide a covering for the class, resulting in the axiom:

$$\text{large-seaport(pn)} \equiv \text{s4(s4.pn)} \vee \text{s5(s5.pn)}$$

By definition, the *head* of an axiom consists of the domain concept together with the set of provided attributes. The *body* of the axiom represents the formula over the sources used

in the axiom. For example, the head of the axiom above is large-seaport(pn), and its body is s4(s4.pn) $\vee$ s5(s5.pn).

Since the number of possible combinations of attributes can be extremely large even for a single class[2] , the mediator does not compute all axioms for all possible combinations of attributes. In a first phase, the system compiles off-line the *minimal set* of domain axioms. Each axiom in the minimal set provides as much information and as many attributes as possible for a particular combination of sources. For example, the previous axiom does not belong to the minimal set because two more attributes, cr and gc, can be obtained from the combination of s4 and s5. Instead, the following axiom is the one included in the minimal set because it incorporates all the attributes for this particular combination of sources:

$$\text{large-seaport(cr gc pn)} \equiv \text{s4(s4.cr s4.gc s4.pn)} \vee \text{s5(s5.cr s5.gc s5.pn)}$$

In the second phase, the system organizes the axioms for each domain class into a lattice. Each lattice is a partial order, using set containment, over subsets of attributes for a domain class.[3] This data structure caches the axioms that have already been computed and makes it possible to efficiently derive, at runtime, the axioms that are relevant for the set of attributes in each particular query. For example, if a query requests large-seaport(gc pn), the corresponding axiom(s) will be derived from the stored axiom(s) for large-seaport(cr gc pn).

Section 3.1 describes the compilation of the minimal domain axioms. Section 3.2 describes some optimization to the compilation process and the axiom projection algorithm. Section 3.3 describes how the lattice structure is constructed and subsequently used during query processing. Appendix A presents some formal properties of the axioms and the compilation algorithm.

3.1. Compiling the minimal set of domain axioms

The system compiles the minimal set of domain axioms by applying a set of five inference rules. Each rule captures an orthogonal type of inference about how sources can be combined based on the features of our representation language. The five rules are:

- **Direct**: Translates the user-provided source definitions into axioms;
- **Covering**: Exploits the covering relationships in the domain model;
- **Definition**: Exploits the constraints in the definition of a domain class;
- **Inherit**: Exploits the inheritance of superclass attributes via shared keys;
- **Compose**: Combines axioms on a given class to provide additional attributes.

Our compilation algorithm first applies the Direct rule, and then the remaining four rules are applied in parallel until quiescence. Our algorithm is incremental, similar in spirit to the semi-naive evaluation of logic programs. In the application of each rule, at least one of the axioms involved must belong to the most recent generation. In order to avoid unnecessary computation, redundant and subsumed axioms are eliminated at each generation. To facilitate this process, the axioms are stored in a normal form (see Section 3.2). The remainder of this section explains each of the rules above based on the example presented in Section 2.2.

seaport(cr gc pn)	≡	**s1(s1.cr s1.gc s1.pn)**	*1.1*
small-seaport(cr pn)	≡	**s6(s6.cr s6.pn)**	*1.2*
large-seaport(gc)	≡	**s3(s3.gc)**	*1.3*
large-seaport(pn)	≡	**s2(s2.pn)**	*1.4*
large-seaport(cn pn)	≡	**s7(s7.cn s7.pn)**	*1.5*
american-large-seaport(cr gc pn)	≡	**s4(s4.cr s4.gc s4.pn)**	*1.6*
european-large-seaport(cr ec gc pn)	≡	**s5(s5.cr s5.ec s5.gc s5.pn)**	*1.7*

Figure 2. Axiom state after application of the direct rule.

The direct rule. The Direct rule installs each source declaration into an axiom for the appropriate class. For instance, the axiom:

$$\text{seaport(cr gc pn)} \equiv \text{s1(s1.cr s1.gc s1.pn)} \wedge \text{cr} = \text{s1.cr} \wedge \text{gc} = \text{s1.gc} \wedge \text{pn} = \text{s1.pn} \tag{1.1}$$

specifies that one way to obtain the number of cranes (cr), geoloc-code (gc), and port-name (pn) of seaport is by using the information source s1. Note that the axiom is expressed in terms of equivalence (≡), not containment; that is, we are declaring that s1(s1.cr s1.gc s1.pn) is co-extensional with seaport(cr gc pn). The series of equality constraints on the right-hand side detail the exact mapping between the domain-level attributes {cr gc pn} and the source attributes {s1.cr s1.gc s1.pn}. Hereafter in the presentation, we will elide these binding equality constraints.

After the execution of the Direct rule, seven axioms are associated with the classes of our sample domain (see figure 2). These axioms correspond precisely to the source definitions depicted in figure 1.

The covering rule. When a class in the domain model is defined as being equivalent to a covering of two or more of its subclasses, we use the covering rule to generate new axioms for the parent class based on the axioms of its subclasses. The rule retrieves the axioms for each subclass and forms the Cartesian product of these sets of axioms. For each tuple of axioms in the Cartesian product, it computes the intersection of the attributes provided by each axiom. If the intersection is not empty, the rule generates a new axiom that provides the common attributes. Then, from each axiom in the tuple, it projects out any attributes not included in the intersection. The body of the new axiom consists of the disjunction of these projected axioms. For example, consider the following covering in the domain model:

$$\text{large-seaport} \equiv \text{american-large-seaport} \vee \text{european-large-seaport}$$

The Covering rule retrieves axiom (1.6) for american-large-seaport, which provides cr, gc, and pn, and axiom (1.7) for european-large-seaport, which provides cr, ec, gc, and pn. The intersection of the attribute sets is {cr, gc, pn}. Both axioms are projected through this

intersection and then combined by disjunction. In this case, the projection amounts simply to removing the attribute ec from the axiom for european-large-seaport. The resulting axiom is:

$$\text{large-seaport(cr gc pn)} \equiv \text{s4(s4.cr s4.gc s4.pn)} \vee \text{s5(s5.cr s5.gc s5.pn)} \quad (2.4)$$

The Covering rule is applied across the class hierarchy in a bottom-up fashion, which ensures that a covering axiom produced at a lower level can participate in a later covering at a higher level. For example, a covering axiom for large-seaport, computed from american-large-seaport and european-large-seaport, is subsequently used in the covering of seaport. The specification of the algorithm used to process the coverings is presented in figure 3. In our example domain, after processing the coverings throughout the entire hierarchy, four more axioms are added, for a total of 11, as shown in figure 4.

The definition rule. This rule exploits the constraints in the definitions of a domain class. Essentially, when a class C is defined in terms of a parent class P and a set of constraints R, the rule generates new axioms for C by conjoining the axioms of P with the constraints R (note that only the axioms of P that provide all attributes used in the constraints R can be used to generate axioms for C). To avoid generating non-minimal axioms, the rule does not consider any parent axiom that includes a source for C.[4] Such situations may occur if

---

$\forall C = C_1 \vee C_2 \vee \cdots \vee C_i \vee \cdots \vee C_n = \bigvee_i C_i$,
where $C \in \mathbf{C}$ ranges from the leaves
to the root of the hierarchy,
and $C_i \in \mathbf{C}$
For each element $\{...[a_{1k}(\bar{x}_{1k}), a_{2l}(\bar{x}_{2l}), ..., a_{nm}(\bar{x}_{nm})]...\}$
$\in A_1 \times \cdots \times A_n$, where $A_i = \text{AXIOMS}(C_i)$
Let $\bar{x} = (\bar{x}_{1k} \cap \bar{x}_{2l} \cap \cdots \cap \bar{x}_{nm})$.
If $\bar{x} \neq \emptyset$ then add to $\text{AXIOMS}(C)$
the new axiom $a(\bar{x}) = \text{PROJECTION}(a(\bar{x}_{1k}), \bar{x}) \vee$
$\text{PROJECTION}(a(\bar{x}_{2k}), \bar{x}) \vee$
$\cdots \vee$
$\text{PROJECTION}(a(\bar{x}_{nm}), \bar{x})$

Notation:

$\mathbf{C}$ is the set of all domain classes in the model hierarchy.
C is a particular domain class, $C \in \mathbf{C}$.
$\text{AXIOMS}(C)$ holds the set of axioms for class C; it can be updated.
$a(\bar{x}) \in \text{AXIOMS}(C)$ is an axiom that provides the attribute set $\bar{x}$.
$\text{PROJECTION}(a(\bar{x}), \bar{y}), \bar{y} \subseteq \bar{x}$ is a subroutine that eliminates from $a(\bar{x})$ all attributes and terms not needed to obtain all of $\bar{y}$ (see the discussion about projection at the end of this section).

Figure 3. Algorithm for the covering rule.

seaport(pn)	≡	**s2(s2.pn) ∨ s6(s6.pn)**	*2.1*
	≡	**s6(s6.pn) ∨ s7(s7.pn)**	*2.2*
seaport(cr pn)	≡	**s4(s4.cr s4.pn) ∨ s5(s5.cr s5.pn) ∨ s6(s6.cr s6.pn)**	*2.3*
seaport(cr gc pn)	≡	s1(s1.cr s1.gc s1.pn)	*1.1*
small-seaport(cr pn)	≡	s6(s6.cr s6.pn)	*1.2*
large-seaport(gc)	≡	s3(s3.gc)	*1.3*
large-seaport(pn)	≡	s2(s2.pn)	*1.4*
large-seaport(cn pn)	≡	s7(s7.cn s7.pn)	*1.5*
large-seaport(cr gc pn)	≡	**s4(s4.cr s4.gc s4.pn) ∨ s5(s5.cr s5.gc s5.pn)**	*2.4*
american-large-seaport(cr gc pn)	≡	s4(s4.cr s4.gc s4.pn)	*1.6*
european-large-seaport(cr ec gc pn)	≡	s5(s5.cr s5.ec s5.gc s5.pn)	*1.7*

Figure 4. Axiom state after application of the covering rule (new axioms in bold).

the axiom of P was generated by a covering that includes C; in this case, the attributes of the axiom of P can only be a subset of those provided by the axioms of C. For example, consider the following definition from the domain model:

$$\text{small-seaport} \equiv \text{seaport} \wedge \text{cr} \leq 7$$

Then, the Definition rule considers axiom (1.1) for seaport that provides the attribute cr needed for the constraint cr $\leq$ 7. Since (1.1) did not result from a covering over small-seaport, the rules yields the following axiom:

$$\text{small-seaport(cr gc pn)} \equiv \text{s1(s1.cr s1.gc s1.pn)} \wedge \text{s1.cr} \leq 7 \tag{3.1}$$

Note that the other three axioms for seaport are built from coverings involving the source s6 for small-seaport. Consequently, they cannot provide more attributes than those directly provided by s6; thus, they are ignored by the definition rule.

The Definition rule is applied top-down across each class in the hierarchy of the domain model. Processing in this order means that a given class may exploit definition axioms created in terms of axioms from its ancestors as well as its direct parent. The specification of the algorithm used to handle class definitions is shown in figure 5. In our example domain, the application of the Definition rule generates two new axioms, for a total of 13, as shown in figure 6.

The inherit rule. When a class C in the domain model shares a key with its ancestor A that has sources for some attributes not available in C, the Inherit rule joins axioms from each class over the shared key in order to provide the new attributes to the descendant. Intuitively, the information about the class A can be transferred to its descendant C as long

$\forall C = C' \wedge c_1 \wedge c_2 \wedge \cdots \wedge c_n, C \in \mathbf{C}, C' \in \mathbf{C},$
where C' is an (immediate) parent class of C and
each constraint $c_i = (b_i\ \theta\ m_i)$
with $b_i \in$ ATTRIBUTES(C_i),
$\theta\ \in \{>, \geq, <, \leq, =, \neq\}$,
and constant m_i
Let $\bar{x} = \{b_i : 1 \leq i \leq n\}$ be the set of all domain attributes used
in all definitional constraints for C
$\forall a'_k(\bar{x}_k) \in$ AXIOMS(C')
if $\bar{x} \subseteq \bar{x}_k$ and a'_k
was not derived from a covering that includes C,
then add axiom $a(\bar{x}_k \cup \bar{x}) = a'_k(\bar{x}_k) \wedge c_1 \wedge c_2 \wedge \cdots c_n$
to AXIOMS(C)

Notation:

C denotes a domain class.
ATTRIBUTES(C) is the set of all attributes defined in the domain class C.
$b \in$ ATTRIBUTES(C) or $b \in \bar{x}$ is a particular domain attribute.
c denotes a constraint expression used in a class definition. Order constraints (*e.g.*, (cr > 7)), equality constraints (*e.g.*, (pn = "Long Beach")) and inequality constraints (*e.g.*, (gc ≠ "XJJD")) between a single attribute and constant are currently supported.
θ is an operator used in a constraint.
m is a constant used in a constraint.

Figure 5. Algorithm for the Definition rule.

as the system has a way of identifying the objects that belong only to the subclass C. In order to do so, the Inherit rule conjoins an ancestor axiom with an axiom of the subclass. For example, as one can see in figure 6 , the class american-large-seaport has no axiom that provides the attribute cn. However, as the axioms (1.6) (for american-large-seaport) and (1.5) (for large-seaport) share the key {pn}, the Inherit rule can generate the following axiom for the class american-large-seaport:

$$\text{american-large-seaport(cn cr gc pn)} \equiv \text{s4(s4.cr s4.gc s4.pn)} \wedge \text{s7(s7.cn s7.pn)} \tag{4.5}$$

The Inherit rule applies bottom-up across the class hierarchy and looks at *all* ancestors of each class. In this way, it considers all possible class/superclass combinations without ever encountering any axioms generated by previous applications of the same rule to intervening classes. The algorithm for the Inherit rule is shown in figure 7. For our example domain, this rule generates six additional axioms, for a total of 19, as shown in figure 8.

The compose rule. The Compose rule ensures that the axioms for a domain class C contain as many attributes as possible given the combinations of relevant sources. For each pair of

seaport(pn)	$\equiv$	s2(s2.pn) $\vee$ s6(s6.pn)	*2.1*
	$\equiv$	s6(s6.pn) $\vee$ s7(s7.pn)	*2.2*
seaport(cr pn)	$\equiv$	s4(s4.cr s4.pn) $\vee$	
		s5(s5.cr s5.pn) $\vee$	
		s6(s6.cr s6.pn)	*2.3*
seaport(cr gc pn)	$\equiv$	s1(s1.cr s1.gc s1.pn)	*1.1*
small-seaport(cr pn)	$\equiv$	s6(s6.cr s6.pn)	*1.2*
small-seaport(cr gc pn)	$\equiv$	**s1(s1.cr s1.gc s1.pn)** $\wedge$	
		s1.cr $\leq$ **7**	*3.1*
large-seaport(gc)	$\equiv$	s3(s3.gc)	*1.3*
large-seaport(pn)	$\equiv$	s2(s2.pn)	*1.4*
large-seaport(cn pn)	$\equiv$	s7(s7.cn s7.pn)	*1.5*
large-seaport(cr gc pn)	$\equiv$	s4(s4.cr s4.gc s4.pn) $\vee$	
		s5(s5.cr s5.gc s5.pn)	*2.4*
	$\equiv$	**s1(s1.cr s1.gc s1.pn)** $\wedge$	
		s1.cr $>$ **7**	*3.2*
american-large-seaport(cr gc pn)	$\equiv$	s4(s4.cr s4.gc s4.pn)	*1.6*
european-large-seaport(cr ec gc pn)	$\equiv$	s5(s5.cr s5.ec s5.gc s5.pn)	*1.7*

Figure 6. Axiom state after application of the Definition rule (new axioms in bold).

$\forall C \in \mathbf{C}$ having ancestor $C' \in \mathbf{C}, \textsc{Keys}(C) \cap \textsc{Keys}(C') \neq \emptyset$,
$\forall a_i(\bar{x}_i) \in \textsc{Axioms}(C)$
$\forall a'_j(\bar{x}'_j) \in \textsc{Axioms}(C')$
where $\bar{x}'_j - \bar{x}_i \neq \emptyset$
If $\exists \bar{k} \subseteq \bar{x}'_j \cap \bar{x}_i \wedge \bar{k} \in \textsc{Keys}(C) \wedge a'_j(\bar{x}'_j)$
was not derived from a covering including C
then add axiom $a_{\bar{k}}(\bar{x}_i \cup \bar{x}'_j) = a_i \wedge a_j$
to $\textsc{Axioms}(C)$.

Notation:

$\textsc{Keys}(C)$ is the attribute or set of attributes that uniquely identify the elements of a class.

Figure 7. Algorithm for the Inherit rule.

C axioms that share a key[5] and provide at least an attribute that the other axiom does not include in its head, the system creates an axiom which pools together the two sets of attributes. For example, axioms (1.5) and (2.4) for large-seaport can be composed because they share the key {pn}, and attributes {cn} and {gc, cr} appear only in (1.5) and (2.4), respectively. The Compose rule conjoins these two axioms, resulting in the following new

seaport(pn)	$\equiv$	s2(s2.pn) $\vee$ s6(s6.pn)	*2.1*
	$\equiv$	s6(s6.pn) $\vee$ s7(s7.pn)	*2.2*
seaport(cr pn)	$\equiv$	s4(s4.cr s4.pn) $\vee$	
		s5(s5.cr s5.pn) $\vee$	
		s6(s6.cr s6.pn)	*2.3*
seaport(cr gc pn)	$\equiv$	s1(s1.cr s1.gc s1.pn)	*1.1*
small-seaport(cr pn)	$\equiv$	s6(s6.cr s6.pn)	*1.2*
small-seaport(cr gc pn)	$\equiv$	s1(s1.cr s1.gc s1.pn) $\wedge$	
		s1.cr ≤ 7	*3.1*
	$\equiv$	**s1(s1.cr s1.gc s1.pn)** $\wedge$	
		s6(s6.cr s6.pn) $\wedge$	
		s1.pn = s6.pn	*4.1*
large-seaport(gc)	$\equiv$	s3(s3.gc)	*1.3*
large-seaport(pn)	$\equiv$	s2(s2.pn)	*1.4*
large-seaport(cn pn)	$\equiv$	s7(s7.cn s7.pn)	*1.5*
large-seaport(cr gc pn)	$\equiv$	s4(s4.cr s4.gc s4.pn) $\vee$	
		s5(s5.cr s5.gc s5.pn)	*2.4*
	$\equiv$	s1(s1.cr s1.gc s1.pn) $\wedge$	
		s1.cr > 7	*3.2*
	$\equiv$	**s1(s1.cr s1.gc s1.pn)** $\wedge$	
		s3(s3.gc) $\wedge$	
		s1.gc = s3.gc	*4.2*
	$\equiv$	**s1(s1.cr s1.gc s1.pn)** $\wedge$	
		s2(s2.pn) $\wedge$	
		s1.pn = s2.pn	*4.3*
large-seaport(cn cr gc pn)	$\equiv$	**s1(s1.cr s1.gc s1.pn)** $\wedge$	
		s7(s7.cn s7.pn) $\wedge$	
		s1.pn = s7.pn	*4.4*
american-large-seaport(cr gc pn)	$\equiv$	s4(s4.cr s4.gc s4.pn)	*1.6*
american-large-seaport(cn cr gc pn)	$\equiv$	**s4(s4.cr s4.gc s4.pn)** $\wedge$	
		s7(s7.cn s7.pn) $\wedge$	
		s4.pn = s7.pn	*4.5*
european-large-seaport(cr ec gc pn)	$\equiv$	s5(s5.cr s5.ec s5.gc s5.pn)	*1.7*
european-large-seaport(cn cr ec gc pn)	$\equiv$	**s5(s5.cr s5.ec s5.gc s5.pn)** $\wedge$	
		s7(s7.cn s7.pn) $\wedge$	
		s5.pn = s7.pn	*4.6*

Figure 8. Axiom state after application of the Inherit rule (new axioms in bold).

axiom (shown in disjunctive normal form):

$$\begin{aligned}&\text{large-seaport(cn cr gc pn)} \equiv \\ &\quad [\text{s4(s4.cr s4.gc s4.pn)} \wedge \text{ s7(s7.cn s7.pn)} \wedge \text{ s4.pn = s7.pn}] \vee \\ &\quad [\text{s5(s5.cr s5.gc s5.pn)} \wedge \text{ s7(s7.cn s7.pn)} \wedge \text{ s5.pn = s7.pn}]\end{aligned} \tag{5.1}$$

$\forall C \in \mathbf{C}$, Construct
HEADS$(C) = \{...\bar{x}_k, \bar{x}_{k+1}...\}$: set of distinct sets of attributes for concept C
COMBOS(C): set of combinations of heads and resulting new head
Initially COMBOS$(C) = \{...\{\bar{x}_k, \{\bar{x}_k\}\}, \{\bar{x}_{k+1}, \{\bar{x}_{k+1}\}\}...\}$
While new combos
$\forall\ \bar{x}_k \in$ HEADS(C)
$\forall (\overline{h}_j, B_j) \in$ COMBOS(C)
If $(\bar{x}_k \cup \overline{h}_j)$ strictly dominates $\overline{h}_j$
and $(\{\bar{x}_k\} \cup B_j)$ is internally non-redundant
then add new combo $\{(\bar{x}_k \cup \overline{h}_j), (\{\bar{x}_k\} \cup B)\}$
to COMBOS(C)
$\forall$ combination $\{\overline{h}_i, \{b_{i1}, b_{i2}, ...\}\} \in$ COMBOS(C)
$\forall$ axiom tuple $[a_{i1j}, a_{i2k}, ...] \in$ AXIOMS$(C(b_{i1})) \times$ AXIOMS$(C(b_{i2})) \times ...$
Add the new axiom $a'(\overline{h}_i) = a_{i1j} \wedge a_{i2k} \wedge ...$
to AXIOMS(C)

Figure 9. Algorithm for the compose rule.

The specification of the algorithm for Compose rule is presented in figure 9. Because this rule processes only a single class at a time, it can be applied over the class hierarchy in an arbitrary order. Figure 10 shows the axioms after the Compose rule is applied. In fact, these axioms represent the minimal set for our example domain. As one can see, the compilation process generated a total of 20 axioms from the seven initial source descriptions for the five domain classes.

3.2. Normalization, projection, and optimizations

Normalization. As we already mentioned, our compilation algorithm first applies the Direct rule, and then the remaining four rules are applied in parallel until quiescence. Due to the incremental nature of this algorithm, various sequences of rule applications may eventually lead to redundant and subsumed axioms (e.g., the Compose rule might independently reconstruct an axiom that had already been computed by previous rules). To avoid unnecessary computation, at each generation, we eliminate all these redundant and subsumed axioms. In order to determine whether or not two computed axioms are equivalent, we keep all axioms in a sorted disjunctive normal form, and we remove replicated predicates and implied order predicates within conjunctions and across disjunctions.

Projection. An axiom provides a set of attributes for a class C. Obviously, it also provides all subsets of these attributes. In practice, it is useful to find the simplified expression of an axiom $a(\bar{x})$ when the query requires only a subset $\bar{y}$ of its attributes. We call the resulting axiom $a'(\bar{y})$ the *projection* of $a(\bar{x})$ on $\bar{y}$. We begin by presenting intuitively some projection

seaport(pn)	$\equiv$	s2(s2.pn) $\vee$ s6(s6.pn)	*2.1*
	$\equiv$	s6(s6.pn) $\vee$ s7(s7.pn)	*2.2*
seaport(cr pn)	$\equiv$	s4(s4.cr s4.pn) $\vee$	
		s5(s5.cr s5.pn) $\vee$	
		s6(s6.cr s6.pn)	*2.3*
seaport(cr gc pn)	$\equiv$	s1(s1.cr s1.gc s1.pn)	*1.1*
small-seaport(cr pn)	$\equiv$	s6(s6.cr s6.pn)	*1.2*
small-seaport(cr gc pn)	$\equiv$	s1(s1.cr s1.gc s1.pn) $\wedge$	
		s1.cr ≤ 7	*3.1*
	$\equiv$	s1(s1.cr s1.gc s1.pn) $\wedge$	
		s6(s6.cr s6.pn) $\wedge$	
		s1.pn = s6.pn	*4.1*
large-seaport(gc)	$\equiv$	s3(s3.gc)	*1.3*
large-seaport(pn)	$\equiv$	s2(s2.pn)	*1.4*
large-seaport(cn pn)	$\equiv$	s7(s7.cn s7.pn)	*1.5*
large-seaport(cr gc pn)	$\equiv$	s4(s4.cr s4.gc s4.pn) $\vee$	
		s5(s5.cr s5.gc s5.pn)	*2.4*
	$\equiv$	s1(s1.cr s1.gc s1.pn) $\wedge$	
		s1.cr > 7	*3.2*
	$\equiv$	s1(s1.cr s1.gc s1.pn) $\wedge$	
		s3(s3.gc) $\wedge$	
		s1.gc = s3.gc	*4.2*
	$\equiv$	s1(s1.cr s1.gc s1.pn) $\wedge$	
		s2(s2.pn) $\wedge$	
		s1.pn = s2.pn	*4.3*
large-seaport(cn cr gc pn)	$\equiv$	s1(s1.cr s1.gc s1.pn) $\wedge$	
		s7(s7.cn s7.pn) $\wedge$	
		s1.pn = s7.pn	*4.4*
	$\equiv$	**[s4(s4.cr s4.gc s4.pn) $\wedge$**	
		s7(s7.cn s7.pn) $\wedge$	
		s4.pn = s7.pn] $\vee$	
		[s5(s5.cr s5.gc s5.pn) $\wedge$	
		s7(s7.cn s7.pn) $\wedge$	
		s5.pn = s7.pn]	*5.1*
american-large-seaport(cr gc pn)	$\equiv$	s4(s4.cr s4.gc s4.pn)	*1.6*
american-large-seaport(cn cr gc pn)	$\equiv$	s4(s4.cr s4.gc s4.pn) $\wedge$	
		s7(s7.cn s7.pn) $\wedge$	
		s4.pn = s7.pn	*4.5*
european-large-seaport(cr ec gc pn)	$\equiv$	s5(s5.cr s5.ec s5.gc s5.pn)	*1.7*
european-large-seaport(cn cr ec gc pn)	$\equiv$	s5(s5.cr s5.ec s5.gc s5.pn) $\wedge$	
		s7(s7.cn s7.pn) $\wedge$	
		s5.pn = s7.pn	*4.6*

Figure 10. Axiom state after application of the Compose rule (new axioms in bold). Minimal set of domain axioms.

examples. Then, we provide a formal definition. Consider the axiom:

$$\begin{aligned}&\text{large-seaport(cn cr gc pn)} \equiv \\ &\quad [\text{s4(s4.cr s4.gc s4.pn)} \wedge \text{s7(s7.cn s7.pn)} \wedge \text{s4.pn=s7.pn}] \vee \\ &\quad [\text{s5(s5.cr s5.gc s5.pn)} \wedge \text{s7(s7.cn s7.pn)} \wedge \text{s5.pn=s7.pn}]\end{aligned} \tag{5.1}$$

The result of the projection of axiom (5.1) on {cr, gc, pn} is:

$$\text{large-seaport(cr gc pn)} \equiv \text{s4(s4.cr s4.gc s4.pn)} \vee \text{s5(s5.cr s5.gc s5.pn)} \tag{5.1$'$}$$

Informally, the reasoning behind this projection is the following. Given our source descriptions, either s7(s7.cn s7.pn) or [s4(s4.cr s4.gc s4.pn) ∨ s5(s5.cr s5.gc s5.pn)] is sufficient to ensure that the set of objects retrieved by (5.1′) is co-extensional with the large-seaport class. As we only need the attributes {cr, gc, pn}, which can be all obtained from the disjunction of s4 and s5, s7 is unnecessary and the projected axiom can be simplified. This example shows that a predicate appears in an axiom essentially because either it contributes some of the needed attributes, or it is a component of the formula that proves that the axiom is equivalent to the given concept. Consider now the axiom (3.2):

$$\text{large-seaport(cr gc pn)} \equiv \text{s1(s1.cr s1.gc s1.pn)} \wedge \text{s1.cr} > 7 \tag{3.2}$$

and its projection on {pn}:

$$\text{large-seaport(pn)} \equiv \text{s1(s1.cr s1.pn)} \wedge \text{s1.cr} > 7 \tag{3.2$'$}$$

In this case, the constraint s1.cr > 7 is needed to ensure that the seaports provided by s1 are indeed large-seaports. In order to enforce this constraint, the attribute s1.cr has to be retrieved from s1 even though it is not one of the requested attributes. Other examples of projection appear in Section 3.3.

Formally, a subformula g of an axiom $a = g \wedge r$ for C ($a \equiv C$) is called a *grounding* of a if $g \equiv C$ and no subformula of g is also a grounding. In other words, g is a minimal subformula of a that entails equivalence to the class C. Our system efficiently computes the groundings of an axiom by using the derivation proof of the axiom. Recursively, the groundings of a compose are the groundings of its components, the groundings of a covering are the disjunction of the groundings of components, the grounding of a definition is the set of constraints, and the grounding of a direct is the direct source itself. For example, axiom (5.1) is obtained by applying the Compose rule to a Covering axiom and a Direct axiom. Therefore, the two groundings are s7(s7.cn s7.pn) (a direct grounding), and [s4(s4.cr s4.gc s4.pn) ∨ s5(s5.cr s5.gc s5.pn)] (a disjunction of direct groundings for each of the subclasses in the covering).

The *projection* of an axiom $a(\bar{x})$ for a class C on the set of attributes $\bar{y}$, $\bar{y} \subseteq \bar{x}$, is an axiom $a'(\bar{y})$ that satisfies:

1. All predicates of a' appear in a
2. a' contains at least one grounding of a

3. All predicates in the conjunctions of a' are connected. Note that connections only occur either between predicates that share a common key, or between a predicate and an order constraint on the attribute used in the order constraint.
4. The only attributes in the body of a' are those needed to satisfy the previous condition or are requested in $\bar{y}$.
5. Each non-grounding predicate in a' must provide some attribute that is not provided by any other predicate.

Optimizations. Some combinations of rule applications can generate axioms that are guaranteed to be redundant. Although these axioms could be removed after the normalization and projection process, we have optimized the speed of the compilation algorithm by including tests that prevent generating redundant axioms:

1. The Covering rule generates an axiom for a class $C = C_1 \vee C_2 \vee \cdots \vee C_n$ only if none of the proposed axioms for C_i contain a source for C or one of its superclasses.
2. The Definition rule generates an axiom for a class $C = C_1 \wedge \mathit{Constraints}$ from an axiom a_1 of its superclass C_1 only if a_1 does not contain a covering involving a source for C.
3. The Inherit rule generates an axiom for a class C based on an axiom a_1 of its superclass C_1 only if a_1 does not contain a covering involving a source for C.
4. The Inherit rule generates an axiom for a class C based on an axiom a_1 of its superclass C_1 and another axiom a of C only if a was not obtained by applying the definition rule to an axiom of C_1.

3.3. Using the domain axioms efficiently

Each axiom in the minimal set provides as many attributes as possible for its particular combination of sources. However, a user query may request any subset of the attributes of a domain class. Therefore, new axioms for the desired attributes have to be efficiently derived from the minimal set. Furthermore, the system needs an effective way to verify whether or not a query is satisfiable (for instance, there might be no sources for the desired attributes). To that effect, the axioms for each class in the domain model are organized in a lattice. Each node in the lattice holds the axioms that represent all alternative ways of obtaining a particular set of attributes. The edges of the lattice capture set containment on attributes.[6] An example of an axiom lattice that contains only the minimal axioms for the class large-seaport appears in figure 11.

A lattice for a domain class is constructed in two phases. In the first phase, the minimal set of axioms is transferred to the lattice and the nodes are completed with *supplementary* axioms. This step is performed off-line, immediately after the compilation of the axioms. In the second phase, the lattice is used as an axiom cache, and new axioms are generated as demanded by user queries. If a node for the requested attributes is already present in the lattice, the axioms of the node are returned. Otherwise, the system creates a new node, populating it with the appropriate axioms. These new axioms are called *interstitial* since they lie between previous axiom sets (both logically and in the lattice).

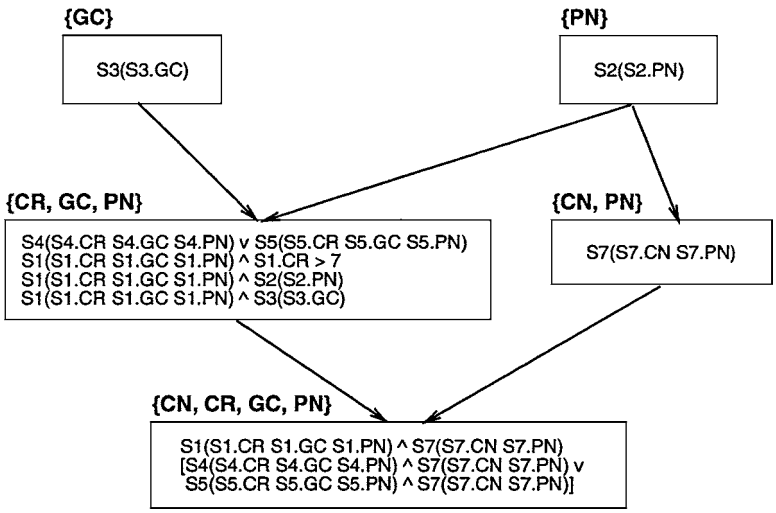

Figure 11. Initial axiom lattice for large-seaport (with minimal axioms only).

Supplementary axioms. The minimal axioms in a node may not list all the possible ways of combining the available sources to provide the attributes of the node. For example, in figure 11, there is one additional way to obtain large-seaport(cr, gc, pn) by combining sources s1 and s7. However, the minimal axiom that combines s1 and s7, (4.4), provides more attributes, {cn, cr, gc, pn}, and it is found in another node of the lattice. Fortunately, the desired axiom is easily computed from (4.4) by projection:

$$\text{large-seaport}(\text{cr},\ \text{gc},\ \text{pn}) \equiv \text{s1}(\text{s1.cr s1.gc s1.pn}) \wedge \text{s7}(\text{s7.pn})$$

Supplementary axioms are computed when the lattice is first constructed. For a class C, the axioms are introduced in its lattice in order of decreasing number of head attributes. This corresponds to filling the diagram in figure 11 bottom up. As each node is added to the lattice, the axioms in the superset (child) nodes are examined. Any projection of a superset axiom into the attributes of the current node is added to the node (unless, of course, the projected axiom is equivalent to any of the existing axioms). The specification of the algorithm for computing the supplementary axioms is shown in figure 12. The supplementary axioms for the example lattice are marked with an asterisk ($*$) in figure 13.

Interstitial axioms. An axiom lattice of a domain class that includes only the minimal and supplementary axioms is usually sparse. A user query may request attributes that are not associated with any node in the lattice. Thus, a new node and axioms will have to be generated. Since the total number of nodes that could be computed is the power set of the attributes, the system does not pre-compile the interstitial axioms. Instead, it derives them on demand from the axioms that already populate the lattice. Consequently, the growth of the lattice results from user queries for unseen sets of attributes.

$\forall C \in \mathbf{C}$
  Create a lattice $\mathcal{L}$ for C.
  Let S be AXIOMS(C) sorted by decreasing number of attributes,
    then lexicographically
  $\forall$ axioms $a(\bar{x}) \in S$
    If a node N providing $\bar{x}$ does not exist in $\mathcal{L}$ then
      create it and link it to any immediate parent and
        children nodes in $\mathcal{L}$
    Add $a(\bar{x})$ to node N.
    $\forall$ children nodes n_c of N
      $\forall a_c \in$ PROJECTION$(n_c, \bar{x})$
        Add a_c to N

Figure 12. Algorithm for determining supplementary axioms.

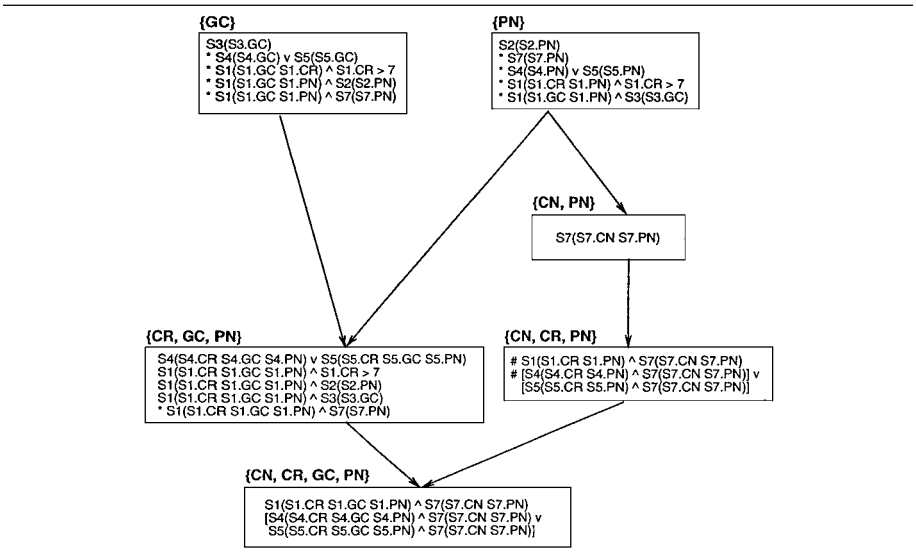

Figure 13. Axiom lattice for large-seaport after generation of all supplementary axioms and single interstitial nodes for large-seaport(cn cr pn). Supplementary axioms are marked with an asterisk (*) and Interstitial axioms are marked with a hash (#).

For example, suppose a query requesting large-seaport(cn cr pn) is received. If a node in the lattice for large-seaport corresponding to the set {cn, cr, pn} exists, the axioms cached in the node are returned. Otherwise, a new node for {cn, cr, pn} will be created and the corresponding interstitial axioms generated. To do so, all nodes that minimally contain the set {cn, cr, pn} become the child nodes of the new node. The axioms from these child nodes are projected into {cn, cr, pn} and used to populate the interstitial node. If no nodes contain

Given some class C and set of attributes $\bar{x}$
Let $\mathcal{L}$ be the lattice for C.
If a node N providing $\bar{x}$ does not exist in $\mathcal{L}$,
    Let N_c be the set of nodes from $\mathcal{L}$ which immediately
        dominate $\bar{x}$(*i.e.*, the children of N)
    If $N_c = \emptyset$
        then fail ($\bar{x}$ cannot be retrieved from the available
                sources of C)
        else create N in $\mathcal{L}$ providing $\bar{x}$
    $\forall n_c \in N_c$
        $\forall a_c \in$ PROJECTION$(n_c, \bar{x})$
            Add a_c to N

Figure 14. Algorithm for computing interstitial axioms.

the requested attribute set, there are no sources available for those attributes and the user query is unsatisfiable. Figure 13 shows the axiom lattice for large-seaport after a request for large-seaport(cn cr pn) has been satisfied, and the interstitial node and axioms (marked with a hash - #) have been added. The algorithm for the synthesis of interstitial axioms is shown in figure 14.

This section has presented an approach to source selection by pre-compiling a set of domain axioms and efficiently using them during query processing. The axioms are used to improve the efficiency of a query planner by having readily available all the alternative ways of obtaining data for domain classes. Moreover, queries for which there are no sources are detected immediately.

4. Binding pattern constraints

Axiom compilation is also applicable when the sources have binding patterns constraints (Ullman, 1989; Kwok and Weld, 1996). A binding pattern constraint is a type of access restriction on a source. More precisely, the constraint requires that some of the tuple's arguments must be bound to constants before producing the rest of the tuple. The role of binding constraints is similar to input parameters of a function: in order to retrieve the output values, one has to provide the input values. Binding constraints are pervasive in Web sources, where, for example, a site that provides stock information may require as input the ticker symbol of a company.

In the previous section, both the axiom generation and the lattice creation were based only on reasoning about the attributes in the heads of the axioms. In the presence of binding patterns, the reasoning process must also use the binding status of each attribute. Intuitively, an attribute with a binding constraint is less general than its unbound version. For example, an axiom with the head seaport(cr gc pn) is more general than seaport(cr gc $pn), where a bound attribute is denoted by prepending the dollar symbol ($) to its name. Axioms with identical heads except for the binding patterns may be incomparable: for example,

seaport(cr gc $pn) and seaport(cr $gc pn) are incomparable, but both of them are more general than seaport(cr $gc $pn).

The algorithms described so far remain essentially the same. Only minimal changes are needed to ensure that binding patterns constraints are respected. In the remaining of the section we highlight the changes and provide some illustrative examples.

4.1. Changes in the rules

Direct rule. We extend our notation for attributes to include binding constraints. For example, if we replace the source s1 from figure 1 with a similar source, s1b, that requires the port-name to be provided in order to return geoloc-code and cranes, we mark the port-name with a dollar ($) symbol. The resulting direct axiom is (cf. axiom (1.1)):

$$\text{seaport(cr gc \$pn)} \equiv \text{s1b(s1b.cr s1b.gc s1b.\$pn)} \tag{1.1$'$}$$

Covering rule. The intersection on the common attributes of the covering is reduced with respect to the binding contraints. For example, if we have the sources s4b(s4b.cr s4b.$gc s4.pn) and s5b(s5b.cr s5b.gc s5b.$pn) (for american-large-seaport and european-large-seaport, respectively), the only union-compatible combination possible is to provide *both* port-name (pn) and geoloc-code (gc) to obtain cranes. The resulting covering axiom is (cf. axiom (2.4)):

$$\begin{aligned}\text{large-seaport(cr \$gc \$pn)} \equiv\ & \text{s4b(s4b.cr s4b.\$gc s4b.pn)} \vee \\ & \text{s5b(s5b.cr s5b.gc s5b.\$pn)}\end{aligned} \tag{2.4$'$}$$

Definition rule. Definitions with equality constraints may satisfy a binding pattern. Thus an axiom derived by definition may have a more general head than the superclass axiom. For example, if we have the domain model definition:

$$\text{american-seaport} \equiv \text{seaport} \wedge \text{cn} = \text{“USA”}$$

and the axiom (cf. axiom (1.1)):

$$\text{seaport(\$cn cr gc \$pn)} \equiv \text{s1c(\$s1c.cn s1c.cr s1c.gc s1c.\$pn)} \tag{1.1$''$}$$

then the resulting axiom for american-seaport is (cf. axiom (3.1)):

$$\begin{aligned}&\text{american-seaport(cn cr gc \$pn)} \equiv \\ &\quad \text{s1c(s1c.cn s1c.cr s1c.gc s1c.\$pn)} \wedge \text{s1c. cn} = \text{“USA”}\end{aligned} \tag{3.1$'$}$$

Inherit and compose rules. The changes affect the conjunction of axioms. A predicate may provide a binding for another predicate, and the resulting axiom may be more general. For example, consider the combination of sources s4b(s4b.cr s4b.$gc s4b.pn) (for american-large-seaport) and s7b(s7b.cn s7b.$pn) (for large-seaport) by the inherit rule.

Once s4b receives a binding for s4b.$gc, it can generate a binding for s4b.pn and pass it to s7b. Therefore, the binding constraint on s7b.$pn is always satisfied internally in the axiom body and the axiom head of the combination is (cn cr $gc pn) as opposed to the simple union of attributes (cn cr $gc $pn). The resulting axiom is (cf. axiom (4.5)):

$$\text{american-large-seaport(cn cr \$gc pn)} \equiv \text{s4b(s4b.cr s4b.\$gc s4.pn)} \wedge \text{s7b(s7b.cn s7b.\$pn)} \quad (4.5')$$

4.2. *Changes in axiom projection*

The five conditions for axiom projection at the end of Section 3.2 have to be completed with a sixth one:

6. There is an ordering of the predicates in the conjunctions of a' that satisfy the binding patterns. A binding constraint is satisfied by either a binding specified in the axiom head or by the attributes of a previous predicate whose constraints are satisfied.

We can compute axioms with binding patterns from axioms without. For example, projecting axiom (5.1) into (cr gc $pn) results in:

$$\text{large-seaport(cr gc \$pn)} \equiv \text{s4(s4.cr s4.gc s4.\$pn)} \vee \text{s5(s5.cr s5.gc s5.\$pn)} \quad (5.1'')$$

However, attributes with binding patterns cannot be projected out. For example, we cannot project axiom (4.5′) with head american-large-seaport(cn cr $gc pn) into american-large-seaport(cn cr pn).

In order to illustrate how binding patterns affect the axiom compilation, let us consider the domain model from figure 15, which was obtained from the one in figure 1 by replacing the original source s4 with s4b. Note that the only difference between the original model and the new one consists of the binding pattern on the attribute gc of s4b (i.e., s4b provides the port name and number of cranes for a *given* geoloc-code). In figure 16 we show the lattice that contains the minimal set of axioms for large-seaport. Compared with the original lattice in figure 11, which displays the minimal set of axioms for large-seaport without binding patterns, the new lattice includes four new axioms, two additional nodes, and four new dominance relationships, which are all shown in bold.

The comparison between the lattices in figures 11 and 16 raises a few interesting points. First of all, the nodes that do not contain s4-based axioms (i.e., {gc}, {pn}, and {cn pn}) remain unchanged. Furthermore, each axiom from figure 11 that does not use s4 also appears in figure 16. Second, the two original axioms in the node {gc cr gc pn} have a different fate: the first one remains unchaged because it does not use s4, while the second one is moved to the newly created node {gc cr $gc pn}. This happens because one of its components, s4b, requires the input paramenter $gc that can not be provided by the other sources in the axiom, and, consequently, $gc becomes an input parameter for the whole axiom. Third, the

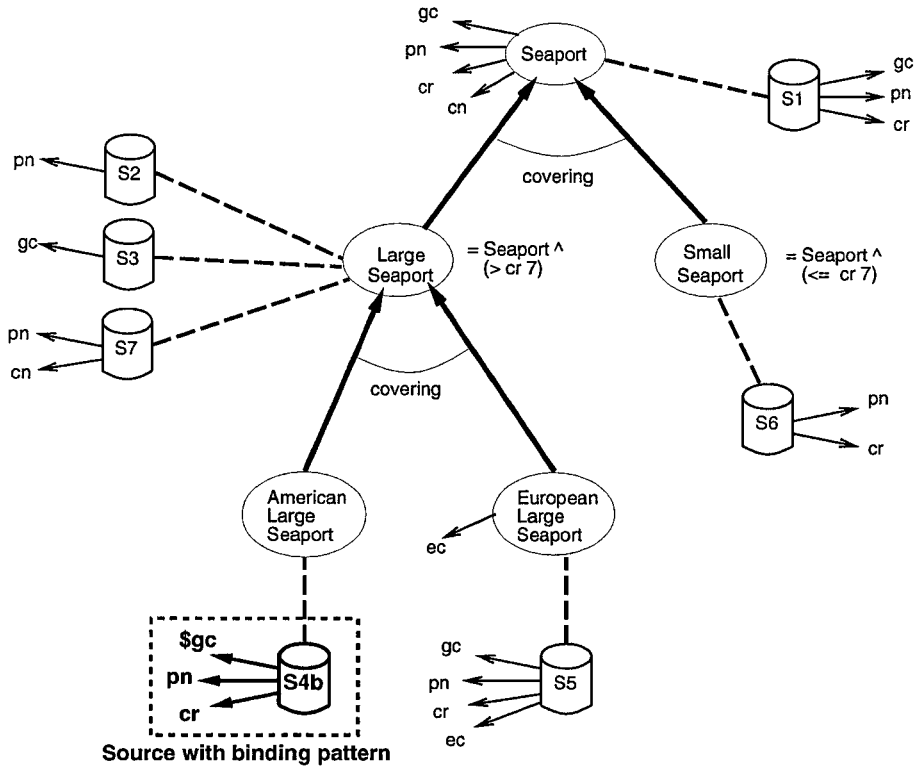

Figure 15. Example domain model and sources with a binding pattern.

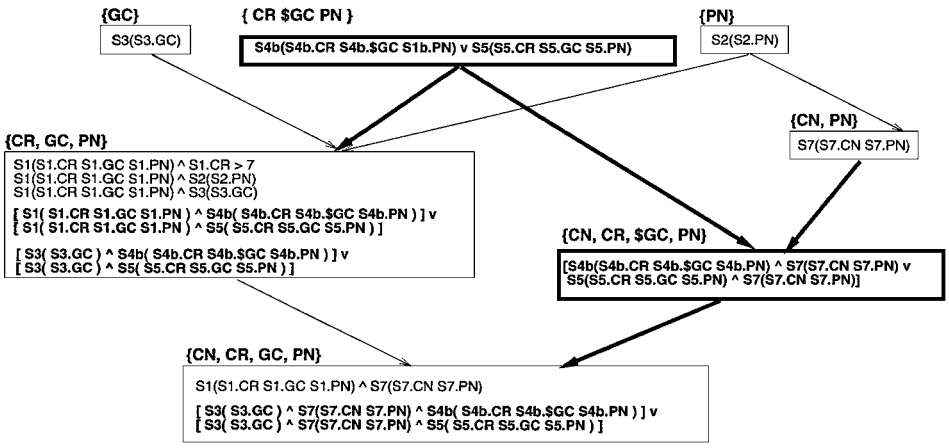

Figure 16. Minimal set of axioms for large-seaport with binding patterns.

node {gc cr gc pn} receives a new axiom:

large-seaport(cn cr gc pn) ≡
    s3(s3.gc) ∧ s7(s7.cn s7.pn) ∧
    [s4b(s4b.cr s4b.$gc s4b.pn) ∨ s5(s5.cr s5.gc s5.pn)]

It is interesting to note the different roles played in the axiom above by s3 and s7: the former is used only to satisfy the binding pattern of s4b, while the latter provides the attribute cn that can not be obtained from the union of s4b and s5. Fourth, the four axioms from the node {cr gc pn} end up in two different nodes. The first axiom, which uses s4, becomes sb4(sb4.cr s4b.$gc s4b.pn) ∨ s5(s5.cr s5.gc s5.pn) and populates the new node {cr $gc pn}. The other three axioms remain unchanged in the {cr gc pn} node. Finally, the same node stores two new axioms:

large-seaport(cr gc pn) ≡
    [s1(s1.cr s1.gc s1.pn) ∧ s4b(s4b.cr s4b.$gc s4b.pn)] ∨
    [s1(s1.cr s1.gc s1.pn) ∧ s5(s5.cr s5.gc s5.pn)]
large-seaport(cr gc pn) ≡
    [s3(s3.gc) ∧ s4b(s4b.cr s4b.$gc s4b.pn)] ∨
    [s3(s3.gc) ∧ s5(s5.cr s5.gc s5.pn)]

Note that the axioms above have the head large-seaport(cr gc pn) even though s4b has a binding pattern on gc. The explanation is straightforward: because of the source ordering in the axiom, one can retrieve values of gc from s1 and s3, respectively, and use them as input arguments for s4b.$gc. As s1 and s3 do not provide new attributes, but rather just satisfy the binding patterns, the axioms above could not have appeared in figure 11 because they would have contained the redundant sources s1 and s3, respectively.

5. Empirical results

This section presents empirical results on the axiom compilation algorithm. We first present results for a number of fielded applications to demonstrate that the algorithm compiles the axioms efficiently for practical real-world domains. We then present results on a set of synthetic domains to evaluate the scaling properties of the algorithms, demonstrating their continued utility as domains grow in size. Both sets of experiments were run on a Sun Ultra 2.

5.1. Experiments in real-world applications

In our first set of experiments, we compiled all axioms in a series of real-world applications built with the SIMS and Ariadne mediators. These applications possess all of the representational features discussed in the paper, generally distributed among several independent hierarchies. However, the density of coverings is low. First, we describe each of the

Table 1. Characteristics of several real-world domains.

Domain	Databases	Sources	Domain classes	Coverings	Binding constr.	Total attr.
Virtual catalog	N/A	6	6	1	6	52
Location finder	N/A	9	7	1	5	46
Entertainment	N/A	10	6	1	14	41
Transportation logistics	8	32	154	0	0	235
Battlefield logistics	6	56	234	0	0	375
Data fusion	12	134	217	1	0	1113
Energy Time series	11	224	435	0	9	4470

Table 2. Axiom compilation in real-world domains.

Domain	Axions	Time (sec.)	Generations
Virtual catalog	10	0.540	3
Location finder	17	0.420	3
Entertainment	17	0.720	4
Transportation logistics	276	34.5	3
Battlefield logistics	293	57.8	3
Data fusion	191	23.0	3
Energy Time series	654	301	5

characteristics of each application domain. Then we show the empirical evaluation of each domain. Characteristic measurements of the complexity and expressiveness of the domains is contained in Table 1. The axiom compilation results for each domain are summarized in Table 2.

The Virtual Catalog application comes from a part supplier domain. The application accesses online electronics part catalogs from Avnet (www.avnet.com), Arrow Electronics (www.arrow.com), and Wyle Electronics (www.wyle.com) by means of Ariadne "wrappers" (Knoblock et al., 2000) which provide a relational view of semi-structured web pages. The application provides price, terms and availability information about parts from different suppliers, given descriptions and/or part numbers. Virtual Catalog uses Ariadne axioms to integrate across the different ways of navigating the individual sites, providing a unified interface.

The Location Finder supports retrieval and integration of country-by-country geographic and political information, based on online sources such as the CIA World Fact Book, the NATO home page, and the United Nations home page. Because the yearly editions of World Fact Book reflects geopolitical changes in the world, a "mapping table" information source is used as well to correlate information from one year to the next.

The Entertainment application combines restaurant directories from online sources CuisineNet (www.cuisinenet.com), Zagat Survey (www.zagat.com), and Fodor's (www.

fodors.com); movie information including theatre listings and movie showings from Yahoo Movies (movies.yahoo.com) and movie trailers (www.hollywood.com); and mapping information from the USGS Tiger map server (tiger.census.gov) and the Etak geocoder (www.geocode .com). All these sources are accessed through Ariadne wrappers that extract the data from the web pages. The application assists a user in browsing the dining and entertainment options in a given city or neighborhood, assembling the results onto a map where clickable icons provide more information. Ariadne axioms link the various information sources (plus computational sources which convert data from one format to another) into a chain which respects their binding patterns and allows the query planner to quickly retrieve the requested information.

The Transportation Logistics application supports general logistical queries about vehicles, their characteristics, capabilities, and requirements, and about geographic locations, ports, passageways, etc., to and through which they might travel. It uses eight ORACLE databases which comprise geography, vehicles and other assets, force readiness information, and unit and apportioned force characteristics. Queries supported in this domain can be quite complex in integrating information across multiple sources, such as "List all ship classes, by ship class name, seaport name, and berth-type name which can handle containers and can dock at Sousse or Tunis, Tunisia."

The Battlefield Logistics application combines relational logistics information similar to that used in the Transportation Logistics above, databases containing military stocked supplied with their locations, and online (web) data sources representing simulations of threat assessment resources and weather forecasts. This application thus integrates conventional and online sources to support allow planning of resource-sensitive military campaigns.

The Data Fusion application uses geographic information including maps and airfield locations, stored both in conventional database and as wrapped web pages, and databases and web pages simulating sensor fusion information, including overlaid weather and map imagery. The application provided an integrated view of the disparate logistics and command information used in a simulated military exercise.

The Energy Time Series application retrieves various measured and computed periodic energy-related measurements, such as retail price, production volume, CPI, etc., from various government agencies, including the Bureau of Labor Statistics, the Energy Information Administration, and the California Energy Commission. These data are stored in over 200 different web pages and relational databases. The model parameterizes and relates each time series, resulting in a large global domain model. This ongoing work also integrates a relational view of SENSUS, a 90,000-concept natural language-derived ontology. This largest Ariadne application to date allows the user to perform cross-agency comparisons not supported by the individual sources.

We detail the parameters of the domains which most affect axiom compilation in Table 1. The table columns detail the size of the domain model, the characteristics of the sources, including the number of coverings and binding constraints between sources. We catalog the axiom compilation results for these domains in Table 2. This table contains the compilation time, resulting axiom counts, and number of generations required until quiescence for the compilation algorithm.

Table 1 shows that the axiom compilation is fast in practice. For example, in the Transportation Logistics domain consisting of a domain model with 154 domain classes with no coverings, and employing 32 sources with no binding constraints between sources distributed over 8 databases, the system compiled 276 axioms in 34.5 seconds, requiring three rounds of rule application before quiescence. Energy Time Series, the largest domain to date, is characterized by a domain model with 435 domain classes and uses 224 sources with 9 binding constraints; compilation time is around five minutes requiring five generations.

In comparison, the example domain we have used throughout the paper generates the 20 axioms from its 5 domain and 6 source classes in less than 0.1 seconds. Note that the example domain, though small, combines all of representational features in one hierarchy.

These results suggest that compilation for domains of realistic size is efficient. Since the algorithm is incremental, even if the domain changes frequently the updates to the axioms can be performed efficiently. More precisely, if one adds a new source to the domain, we do *not* have to recompute all the axioms. Instead, we can add the new source to the existing axioms and let the algorithm run until quiescence (the only potential drawback of such method is that it may not produce the minimal number of axioms). In case of deleting a source, the current version of the algorithm must start from scratch. However, we believe that it is possible to create an enhanced version of our algorithm that checks each axiom and removes/re-writes the ones in which the deleted source appears.

5.2. Experiments on synthetic domains

We also present empirical results on the performance of our axiom compilation algorithm on a set of synthetic domains. The purpose of these experiments is to show the scaling behavior as domain size and complexity are varied. In a series of parameterized experiments in each domain, we test the ability of the axiom compilation algorithm to scale as the number of sources grow.

Each domain in our space of domain configurations consists of one or more hierarchies of classes. The size of this hierarchy ranges from 1 to 5 domain classes. Each class in a hierarchy possesses the same set of attributes, not all of which are provided in any one source. In the single class configuration, only the root domain class is available. Subsequent larger configurations grow the domain downward, providing more domain classes and a richer set of class relationships. In the three-class configuration, two subclasses to the root are added. Both subclasses are defined relative to the parent class and are declared to cover the parent class. In this configuration we thus have two definitions and one covering. In the five-class configuration, one of the subclasses from the three-class configuration above is further extended in the same way. The net tally of this configuration is four definitions and two coverings.

In all of the experiments on the synthetic domains, we randomly generated sources and assigned them in a round-robin fashion to the domain classes. For each source, a set of five attributes were randomly selected plus one shared attribute that serves as the primary key. We ran the axiom compilation algorithm three times for each point and averaged the results.

The first experiment tests the algorithm on a single-hierarchy synthetic domain. The graphs in figure 17 show the run time in CPU seconds and number of axioms graphed

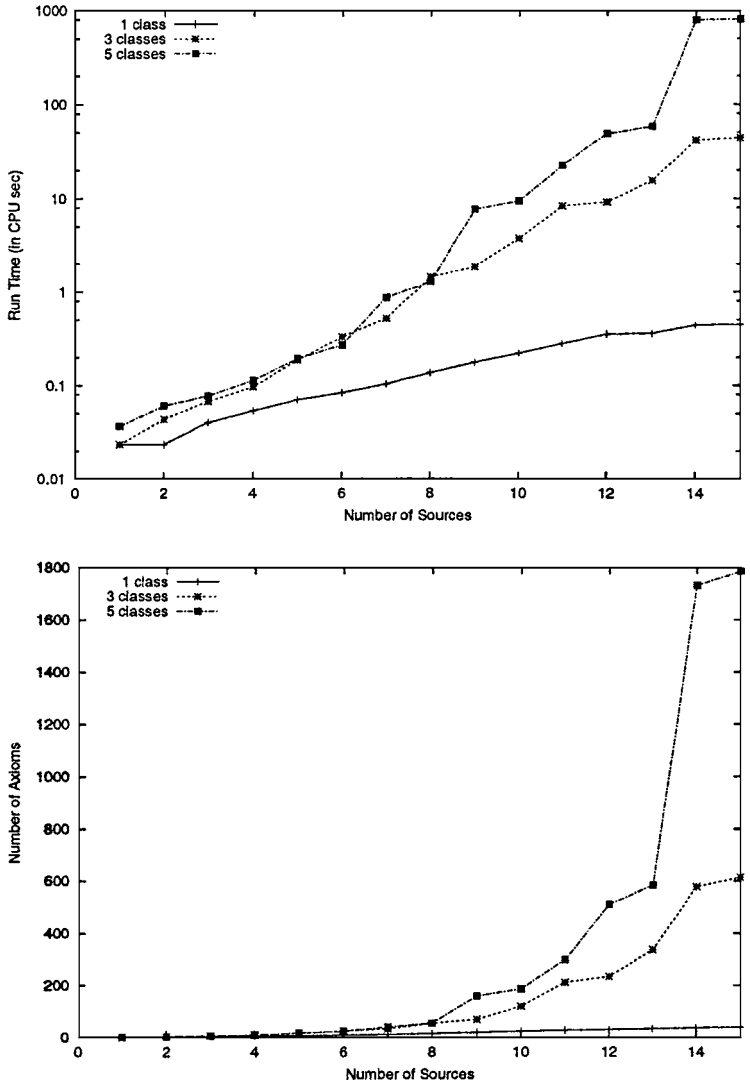

Figure 17. Experiment with a single hierarchy with coverings.

against the total number of sources. In the case of a single class in the hierarchy the number of axioms is close to linear in the number of sources because only the compose rule is applicable and the combination of sources quickly covers the set of attributes. In the case of three and five classes in the hierarchy, the algorithm can easily handle up to 15 related sources. Above that the number of axioms grows quite large. As shown later in this section, the algorithm can scale to much larger numbers of sources if there are multiple hierarchies, which is the typical situation. The reason for the growth in the number of axioms is due to the interaction between the covering and compose rules, which interact to produce an

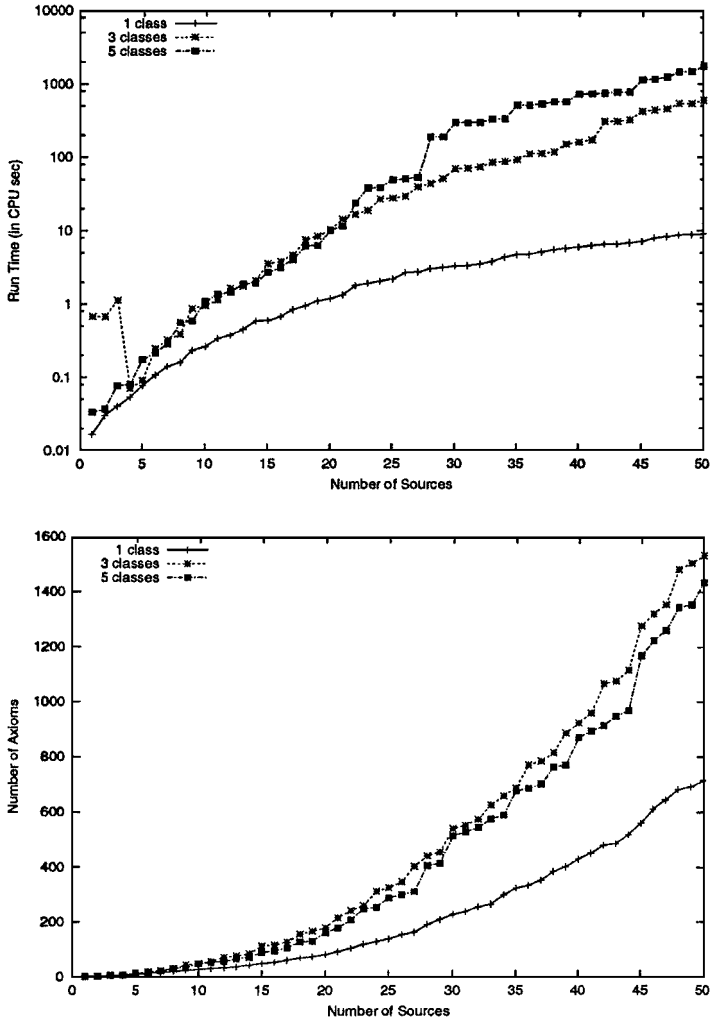

Figure 18. Experiment with a single hierarchy without coverings.

exponential number of combinations. As shown in the next experiment, this can be addressed by limiting the use of the covering rule. Also, we showed on the real-world applications of Section 5.1 that this interaction does not seem to be a problem in practice.

We ran a second experiment, shown in figure 18, that was identical to the first except that there were no coverings in the domain model. Without any coverings, the algorithm is able to scale to 50 or more closely related sources.

Finally, we ran a third experiment, shown in figure 19, on the synthetic domain where instead of a single hierarchy of classes, we started with 10 separate hierarchies and assigned the sources in a round-robin fashion to each of the hierarchies. The purpose of

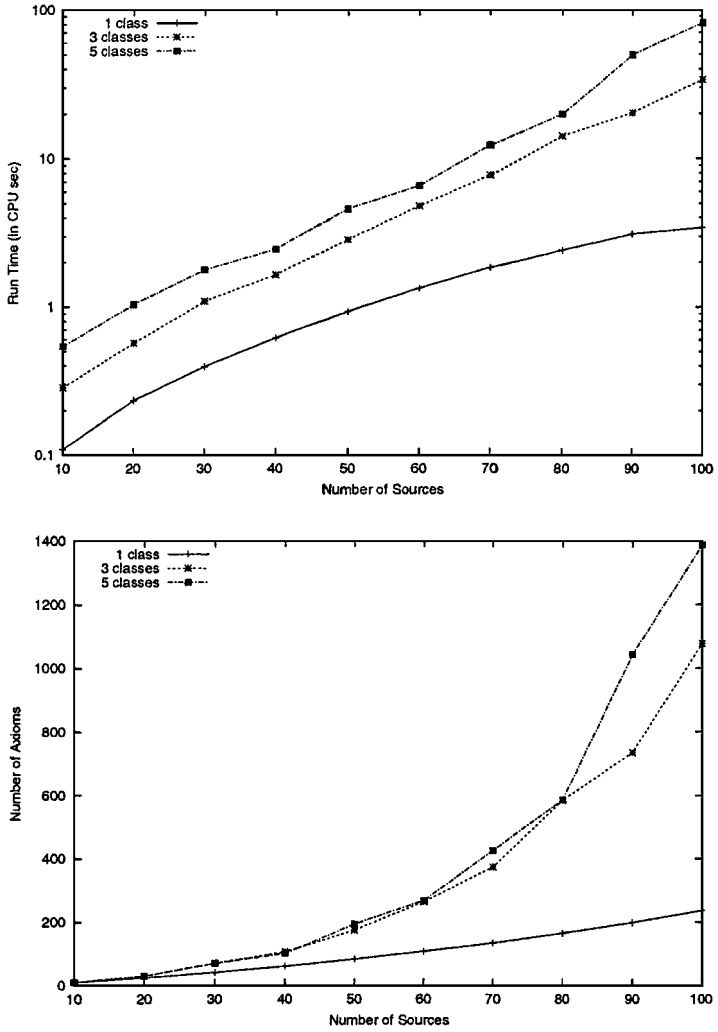

Figure 19. Experiment with multiple hierarchies with coverings.

this experiment is to capture the more realistic situation where not all sources are related to all other sources. Not surprisingly, the compilation algorithm is able to scale easily to 100 sources with a run time that is under 100 CPU seconds. This experiment includes the same coverings that were used in the first experiment.

5.3. Discussion of experiments

The experiments on the real-world applications provide evidence that our approach to axiom compilation does work well in practice. Many of these domains are large complicated domains with axioms that would be difficult to construct manually. The representation

language that we provide is sufficient to describe the sources and the results show that we can then efficiently compile the axioms.

As shown in the experiments on the synthetic domains, the algorithm is clearly exponential. The number of classes in the experiments is small, but the density of sources is quite high within these classes. The algorithm could scale to much larger number of classes if the number of sources per class were not as large, which is the usual case. The experiments also show that the algorithm scales much better without coverings or with minimal use of coverings. In addition, the way in which the axiom compilation is typically used in practice (as shown in the real-world application section) is where there are a number of sources spread over multiple concept hierarchies, which we show makes it possible to scale to hundreds of sources.

There are a number of simple improvements to the algorithms that would allow us to scale to even larger number of sources. First, instead of compiling axioms at the source level, we could compile the axioms in domain terms and then instantiate the domain-level axioms at run-time. This would greatly reduce the number of axioms, which would allow the system to scale to many more sources in the cases where there are many overlapping and redundant sources. The reduction in axioms comes from the fact that some distinctions between the axioms are not relevant at the domain level. Second, a natural extension of this work is to build an incremental version of the algorithm. This would allow the system to build the axioms over time and incrementally add and delete new sources.

6. Related work

Heterogeneous multidatabase systems typically use a global domain model to provide the "glue" to integrate multiple data sources. The global domain model can be seen as providing common semantics to the information sources by means of views that relate terms in the sources with terms in the domain model. There are two ways of specifying these views, which have complementary properties (Ullman, 1997). The first approach, which has been widely used, is to integrate the sources by defining the global schema as a collection of views (queries) over the sources. This general approach has been used in a variety of systems, including Multibase (Landers and Rosenberg, 1982), Pegasus (Ahmed et al., 1991), TSIMMIS (Hammer et al., 1995), and HERMES (Adali et al., 1996). An advantage is that the query rewriting algorithms are very efficient. The rewriting consists of substituting domain terms by their definitions, and simplifying the resulting source-level queries. A disadvantage is that adding or modifying a source can be quite difficult: all definitions in which that source appears have to be manually reconsidered.

The second approach, exemplified by the Information Manifold (Levy et al., 1996b), is to define each data source as a view of the global domain model. Specifically, each source predicate is defined as a view over domain predicates. An advantage of this method is that modifying the definitions or adding new sources is quite straightforward because sources are defined independently from each other. A disadvantage is that the algorithms to rewrite a domain-level query into a source-level query involves testing containment of views, which is computationally expensive (Levy et al., 1996a, 1996b). The work presented in this paper combines the best of both approaches. Initially, sources are conveniently defined in domain

terms, so that new sources can be easily incorporated or updated. Then, axioms defining domain terms as views over source terms are compiled, so that query processing can be performed more efficiently.

Duschka (Duschka, 1997; Duschka and Genesereth, 1997; Duschka and Levy, 1997; Abiteboul and Duschka, 1998) developed an elegant approach to efficiently process queries on views by inverting the axioms that describe the sources and provided an analysis of the data complexity of computing the *certain answers* given a query, view definitions, and view instances. He shows that the certain answers can be computed in polynomial time for several source and query languages. Unfortunately, he shows that for more complex languages (for example, that include disjunction in the view definitions, such as the one presented in this paper), the complexity of computing the certain answers is co-NP hard or worse, depending on the details of the language. In the work presented here, we make a closed-world assumption on the content of the data sources, and our source and query languages correspond to Duschka's class of positive queries and positive views with order constraints. As opposed to Duschka, we precompile the axioms to avoid the expensive search performed at run-time. In addition, we have a hierarchical domain model, which allows us to state relationships between classes, such as inheritance and definitions of classes in terms of other classes. Further analysis of the complexity of certain and possible answers appears in Grahne and Mendelzon (1999).

Another approach that attempts to combine some of the features of both approaches is illustrated by the Garlic system (Haas et al., 1997; Roth and Schwarz, 1997). Instead of defining views of the sources or of the domain model, Garlic maintains a list of the sources that can provide a portion of the data for each class; when Garlic receives a query, it queries in turn each of the possibly relevant sources to determine which portion of the required data the sources can provide. An advantage of this approach is that individual sources can provide accurate estimates of the cost of retrieving the data, so Garlic can put together efficient plans. The work on Garlic addresses a different problem than the one addressed in this paper and could be usefully combined with our approach. Our work focuses on providing a rich representation language for describing the contents of sources and efficiently determining how those sources can be combined to answer a query. In contrast, Garlic focuses on finding the most efficient combination of possible sources, assuming a simple representation language. The two approaches could be combined by first using our work to represent the sources and determine how to combine the relevant sources and then using the Garlic approach to estimate the cost of the different plans and select the most efficient one.

The Information Manifold (IM) (Levy et al., 1996a, 1996b) provides a rich representation language that is combination of datalog and description logic. However, view rewriting in its language is intractable and can become undecidable in the presence of recursion (Levy and Rousset, 1996a, 1996b). Nevertheless, Levy argues that his algorithm focuses on the relevant information sources, which in practice is usually small; consequently, the size of typical queries will also be small, so the theoretical intractability does not represent a major problem. An analysis of some tractable cases in the computation of query containment is given by Saraiya (1991) and Chekuri and Rajaraman (1997). Views in IM express containment relationships, and IM produces query plans that are maximally contained rewritings. This

implies that the system may return an incomplete answer without being able to signal it. By contrast, our view definitions express equality and our query plans provide complete answers. We can straightforwardly determine when a user query cannot be answered. More importantly, we can easily identify the classes with missing information, and, consequently, we can guide the query relaxation.

Recently, in the context of the Esprit project Foundations of Data Warehouse Quality (DWQ), there has been a renewed interest in the theoretical analysis and application of description logics to information integration. A complexity analysis of answering queries using views in description logics appears in Calvanese et al. (1999). An early study of description logics in information integration is (Catarci and Lenzerini, 1993). The DWQ project has also produced analyses of query rewriting in languages with regular expressions (Calvanese et al., 2000) and aggregates (Nutt et al., 1998).

Perhaps the most general approach to information integration is given by context logic (Buvač, 1996; Guha, 1991), which extends the predicate calculus with a new modality, `(ist c` ϕ`)`, meaning that a logical sentence ϕ is true in a context `c`. The logic is sound and complete, but undecidable. Lifting axioms relate formulas in different contexts, similarly to the view definitions above. The Carnot system (Collet et al., 1991) and its successor InfoSleuth (Bayardo et al., 1997) use restricted forms of lifting axioms, similar to those of the Multibase approach, which are expressed in a frame-based common language. A system using full context logic is described in Farquhar et al. (1995). Rather than focusing solely on generality, our work also considers efficient query planning techniques, such as the domain precompilation presented in this paper, while at the same being able to represent a great variety of sources in practice.

7. Discussion

We have presented an approach to integrating information from heterogeneous data sources that combines the *flexibility* of view rewriting with the *efficiency* of query processing typical of systems such as Multibase and TSIMMIS. In order to do so, our system allows the user to conveniently define the information sources in terms of the domain model, and it compiles these source descriptions into a set of axioms that specify the domain model classes as formulas in source terms. Based on the axioms compiled off-line, the system computes at run-time the most appropriate rewriting for answering a query by simply instantiating the corresponding axioms. Our central idea is to shift the complexity of view rewriting to a preprocessing step that is performed off-line and is amortized as the mediator processes all the user queries.

A limitation of our approach is the fact that our representation language does not allow specifying a source as an arbitrary join over the domain classes. The difficulty in supporting this capability is that the domain designer would need to know all possible queries in advance in order to compile a set of axioms that do not lose completeness. Consider a source $S(y, z)$ that is described as $D_1(x, y) \wedge D_2(x, z)$. Even if this source cannot be included in the axioms for D_1 or D_2 it could still be used in a query for $D_1 \wedge D_2$. Therefore, without compiling all possible combinations of domain classes, the axiom compilation would not be complete.

Despite this limitation, there are many applications where our approach is quite useful. Consider the case of building an integrated catalog of electronic parts where the data comes from individual supplier catalogs. In an application like this each supplier is providing different classes of parts, where the top level concept is the class of parts and there are a number of related classes that all restrict the class of parts in various ways. Our axiom compilation approach is ideally suited to applications such as these since the algorithm can compile the complete set of axioms in advance and quickly answer requests using the relevant catalogs.

Our compiled axioms facilitate query processing in the presence of *partial information*. The modeling language allows the user to specify when a source has complete information for a class. However, in many real-world domains complete information will not be available, and the queries will have to be answered based on the available partial information. Having axioms compiled for each domain class allows us to immediately recognize unsatisfiable queries, to identify exactly what class of information is missing, and to inform the user about the information that can be provided.

The compiled axioms also facilitate *replanning after failure*: as information sources in a distributed heterogeneous environment may become unavailable during the execution of a query, it is desirable to provide an alternative way to answer that query, reusing parts of the executed plan if possible. These alternative ways of obtaining the required data are already available in the compiled axioms.

Appendix: Valid axioms

In this section we give a formal description of the kinds of integration axioms that are valid according to the given source and domain models that the SIMS mediator accepts.

Intuitively, the valid integration axioms are those that involve sources that can be arranged in a definitional hierarchy. So the type of source classes that our language accepts have to be related among each other by either being co-extensional or one being contained in another.

Definition 1 (Integration Axiom). An integration axiom has the form $D(\bar{x}) \equiv \phi(\bar{y})$, where D is a domain class, $\bar{x}$ and $\bar{y}$ are sets of attributes ($\bar{x} \subseteq \bar{y}$), ϕ is a formula over source classes with conjunction, disjunction, and order constraints (of the type: attribute θ constant, where $\theta \in \{=, <, \leq, >, \geq\}$). The *head* of the axiom is $D(\bar{x})$ and the *body* is $\phi(\bar{y})$. Without loss of generality assume the axiom body is written in Disjunctive Normal Form (DNF).

Definition 2 (Hierarchy-Supported). Consider a domain hierarchy H and the macro-expansion operator M that takes a definition $D_k \equiv \phi^k(\{D_i\})$ in H and substitutes one of the domain classes D_i of $\phi^k(\{D_i\})$ by its respective definition $D_i \equiv \phi^i(\{D_j\})$); that is, $D_k \equiv \phi^k(\{\ldots, D_{i-1}, \phi^i(\{D_j\}), D_{i+1}, \ldots\})$.

A *domain* formula is hierarchy-supported iff it belongs to the fixpoint of M(H) (*definition*, *covering*).

A *source* formula is hierarchy-supported iff it is obtained from a hierarchy-supported *domain* formula where each domain class D has been substituted by either:

- a source class S, if source description $S(\bar{x}) \equiv D(\bar{x})$ exists (*direct*).
- a conjunction of source classes $S_i \wedge S_j$, if source descriptions $S_i(\bar{x}) \equiv D(\bar{x})$ and $S_j(\bar{y}) \equiv D(\bar{y})$ exist and $\bar{x} \cap \bar{y} \neq \emptyset$ and $\bar{x} \not\subset \bar{y}$ and $\bar{y} \not\subset \bar{x}$ (*compose*).
- a conjunction of source classes $S_i \wedge S_j$, if source descriptions $S_i(\bar{x}) \equiv D_i(\bar{x})$ and $S_j(\bar{y}) \equiv D_j(\bar{y})$ exist and $D_i \sqsubset D_j$ and $\bar{x} \cap \bar{y} \neq \emptyset$ and $\bar{x} \not\subset \bar{y}$ and $\bar{y} \not\subset \bar{x}$ (*inherit*).

Definition 3 (Grounding). A subformula g of an axiom $D(\bar{x}) \equiv \phi(\bar{y})$ is called *grounding* iff $g \equiv D$ and no subformula of g is also a grounding. In other words, g is a minimal subformula of the axiom body that is equivalent to the concept in the axiom head according to the source descriptions and the hierarchy definitions.

Definition 4 (Key-connected). A conjunctive formula is key-connected iff *all* the classes in the formula can be joined on their respective keys. That is, the graph whose nodes are the class names and whose edges represent joins on keys of two classes in the formula is connected.

Definition 5 (Constraint-Safe). A conjunctive formula is constraint-safe iff all the attributes in the constraints are also present in classes of the axiom body, i.e., all constraints in the formula can be evaluated.

Definition 6 (Binding-Safe). A conjunctive formula is binding-safe iff there exists an ordering of the classes such that the binding patterns of each class are satisfied by previous classes.

Definition 7 (Union-Compatible Axiom). An axiom (whose body is in DNF) is union-compatible iff the intersection of the attribute sets of each disjunct is equal to the set of attributes in the axiom head. (If the axiom body is conjunctive then the axiom is trivially union-compatible.)

Definition 8 (Source-Minimal). An axiom is source-minimal iff each predicate in the axiom body satisfies:

- the predicate is a source class and it uniquely provides an attribute (i.e., the attribute does not appear anywhere else in the axiom body and the attribute is present in the axiom head).
- if the predicate is a source class and it does not uniquely provide some attribute or if the predicate is a constraint, then the predicate must participate in any grounding for the axiom (i.e., there must not exist an alternative grounding that does not use this predicate).

Definition 9 (Valid Axiom). An integration axiom is valid iff all of the following conditions are satisfied:

- The axiom body is hierarchy-supported.
- The axiom body contains a grounding.
- Each conjunction in the axiom body is constraint-safe.

- Each conjunction in the axiom body is key-connected.
- Each conjunction in the axiom body is binding-safe (when considering the set of attributes bound in the axiom head as the first element of the binding pattern ordering).
- The axiom is union-compatible.
- The axiom is source-minimal.

Definition 10 (Attribute-Maximal). A valid axiom is attribute-maximal iff each source class in the axiom body provides as many attributes as possible and the axiom still remains valid.

Note that axioms resulting from the axiom projection algorithm described in Section 3.2 are not attribute-maximal but are valid.

Theorem 1 (*Attribute-Maximal Valid Axioms*)**.** *The axiom compilation algorithm of Section* 3.1 *is sound and complete with respect to the computation of attribute-maximal valid axioms for a given domain hierarchy* H *and source descriptions* $S_i(\bar{x}) \equiv D_j(\bar{x})$.

Proof sketch. By construction, the axiom compilation rules compute the hierarchy-supported source formulas (definition 2). The macro-expansion of the hierarchy definitions is achieved by the *definition* and *covering* rules, the substitution of all possible source combinations into the domain-level definitions is accomplished by the *direct*, *inherit*, and *compose* rules. The fixpoint is achieved as all rules are run in parallel until quiescence. Each axiom proposed in each iteration of the rules is checked to be valid and attribute-maximal using the conditions in definitions 9 and 10. □

Definition 11 (Minimal Axiom Set). We are interested primarily in the attribute-maximal valid axioms, since the rest of valid axioms can be derived by axiom projection from them. We define the minimal axiom set to be the set of attribute-maximal valid axioms for a given domain hierarchy H and source descriptions $S_i(\bar{x}) \equiv D_j(\bar{x})$.

Corollary 1 (*Minimal Axiom Set*)**.** *The axiom compilation algorithm of Section* 3.1 *computes the minimal axiom set for a given domain hierarchy* H *and source descriptions* $S_i(\bar{x}) \equiv D_j(\bar{x})$.

Proof: By construction, our algorithm computes the attribute-maximal valid axioms. The same axiom could be obtained by different derivations, but our algorithm filters the axioms using axiom equivalence (cf. Section 3.2), so that only one copy of the axiom is stored.

Acknowledgments

The research reported here was supported in part by the Rome Laboratory of the Air Force Systems Command and the Defense Advanced Research Projects Agency (DARPA) under contract numbers F30602-98-2-0109, F30602-97-2-0352, and F30602-97-2-0238, in part by the United States Air Force under contract number F49620-98-1-0046, and in part by

the Integrated Media Systems Center, a NSF Engineering Research Center. The views and conclusions contained in this article are the authors' and should not be interpreted as representing the official opinion or policy of any of the above organizations or any person connected with them.

Notes

1. For clarity of exposition, a source attribute will be named s.a when it maps to a domain attribute a. However, in our system source and domain attributes may have arbitrary names.
2. The number of *possible* axiom heads is the power set of the set of attributes of a domain concept (i.e., 2^n for n attributes). If we also take into account binding patterns (see Section 4) the number of possible heads becomes 3^n.
3. In the presence of binding patterns set containment is not enough; see Section 4.
4. In our system this type of reasoning is efficient. As axioms are generated from particular rules, it is a simple matter to record the proof tree for each generated axiom. This proof tree can be examined to answer such questions. The details of this proof tree representation have been omitted for brevity.
5. More precisely, for which a *path* of keys can be constructed. A domain concept may have several alternative keys and sources may provide different keys. Nevertheless, the compose rule ensures that all the sources involved in an axiom can be joined together. For example, s1(k1 a) $\wedge$ s2(k1 k2 b) $\wedge$ s3(k2 c) is a valid axiom, but s1(k1 a) $\wedge$ s2(k1 b) $\wedge$ s3(k2 c) is not, because objects from s3 cannot be related to the corresponding objects of s1 or s2.
6. See the extensions for binding patterns in Section 4.

References

Abiteboul, S. and Duschka, O.M. (1998). Answering Queries Using Materialized Views. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. (pp. 254–263), Seattle, WA.

Adali, S., Candan, K.S., Papkonstantinou, Y., and Subrahmanian, V.S. (1996). Query Caching and Optimization in Distributed Mediator Systems. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 25(2), 137–148.

Ahmed, R., Smedt, P.D., Du, W., Kent, W., Ketabchi, M.A., Litwin, W.A., Rafii, A., and Shan, M.C. (1991). The Pegasus Heterogenous Multidatabase System. *IEEE Computer*, pp. 19–27.

Ambite, J.L. and Knoblock, C.A. (2000). Flexible and Scalable Cost-Based Query Planning in Mediators: A Transformational Approach. *Artificial Intelligence Journal*, 118(1-2), 115–161.

Arens, Y., Chee, C.Y., Hsu, C.-N., and Knoblock, C.A. (1993). Retrieving and Integrating Data from Multiple Information Sources. *International Journal on Intelligent and Cooperative Information Systems*, 2(2), 127–158.

Arens, Y., Knoblock, C.A., and Shen, W.-M. (1996). Query Reformulation for Dynamic Information Integration. *Journal of Intelligent Information Systems, Special Issue on Intelligent Information Integration*, 6(2/3), 99–130.

Bayardo, R.J., Bohrer, B., Brice, R.S., Cichocki, A., Fowler, J., Helal, A., Kashyap, V., Ksiezyk, T., Martin, G., Nodine, M., Rashid, M., Rusinkiewicz, M., Shea, R., Unnikrishnan, C., Unruh, A., and Woelk, D. (1997). Info-Sleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Environments. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. (pp. 195–206), Tucson, Arizona.

Brachman, R. and Schmolze, J. (1985). An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, 9(2), 171–216.

Buvač, S. (1996). Quantificational Logic of Context. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 600–606.

Calvanese, D., Giacomo, G.D., and Lenzerini, M. (1999). Answering Queries Using Views in Description Logics. In *Proceedings of the 1999 Description Logic Workshop*, pp. 9–13.

Calvanese, D., Giacomo, G.D., Lenzerini, M., and Vardi, M.Y. (2000). Answering Regular Path Queries Using Views. In *Proceedings of the 16th IEEE International Conference on Data Engineering*. (pp. 389–398), San Diego, CA.

Catarci, T. and Lenzerini, M. (1993). Interschema Knowledge in Cooperative Information Systems. In G.S.M. Huhns and M.P. Papazoglou (Ed.) *Proceedings of the International Conference on Intelligent and Cooperative Information Systems*, (pp. 55–63) Rotterdam, the Netherlands: IEEE Computer Society Press.

Chekuri, C. and Rajaraman, A. (1997). Conjunctive Query Containment Revisited. In *Proceedings of the Sixth International Conference on Database Theory*. (pp. 56–70), Delphi, Greece.

Collet, C., Huhns, M.N., and Shen, W.-M. (1991). Resource Integration Using a Large Knowledge Base in Carnot. *IEEE Computer* pp. 55–62.

Duschka, O.M. (1997). Query Planning and Optimization in Information Integration. Ph.D. Thesis, Stanford University, Department of Computer Science.

Duschka, O.M. and Genesereth, M.R. (1997). Answering Recursive Queries Using Views. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. (pp. 109–116), Tucson, Arizona.

Duschka, O.M. and Levy, A.Y. (1997). Recursive Plans for Information Gathering. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*. (pp. 778–784), Nagoya, Japan.

Farquhar, A., Dappert, A., Fikes, R., and Pratt, W. (1995). Integrating Information Sources Using Context Logic. In *AAAI Spring Symposium on Information Gathering from Distributed Heterogeneous Environments*.

Grahne, G. and Mendelzon, A.O. (1999). Tableau Techniques for Querying Information Sources through Global Schemas. In *7th International Conference on Database Theory*, (pp. 332–347). Jerusalem, Israel.

Guha, R. (1991). Contexts: A Formalization and Some Applications. Ph.D. Thesis, Stanford University.

Haas, L.M., Kossmann, D., Wimmers, E.L., and Yang, J. (1997). Optimizing Queries Across Diverse Data Sources. In *Proceedings of the Twenty-third International Conference on Very Large Data Bases*. (pp. 276–285), Athens, Greece.

Hammer, J., Garcia-Molina, H., Ireland, K., Papakonstantinou, Y., Ullman, J., and Widom, J. (1995). Information Translation, Mediation, and Mosaic-Based Browsing in the TSIMMIS System. In *Proceedings of the ACM SIGMOD Internation Conference on Management of Data*. (p. 483), San Jose, CA.

Knoblock, C.A., Minton, S., Ambite, J.L., Ashish, N., Modi, P.J., Muslea, I., Philpot, A.G., and Tejada, S. (1998). Modeling Web Sources for Information Integration. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. (pp. 211–218), Madison, WI.

Knoblock, C.A., Minton, S., Ambite, J.L., Ashish, N., Muslea, I., Philpot, A.G., and Tejada, S. (2000). The ARIADNE Approach to Web-based Information Integration. *International the Journal on Intelligent Cooperative Information Systems (IJCIS) Special Issue on Intelligent Information Agents: Theory and Applications*, pp. 145–169.

Kwok, C.T. and Weld, D.S. (1996). Planning to Gather Information. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*. Portland, OR.

Landers, T. and Rosenberg, R.L. (1982). An Overview of Multibase. In H. Schneider (Ed.), *Distributed Data Bases*. North-Holland, pp. 153–184.

Levy, A.Y., Mendelzon, A.O., Sagiv, Y., and Srivastava, D. (1995a). Answering Queries Using Views. In *Proceedings of the 14th ACM Symposium on Principles of Database Systems*. (pp. 95–104), San Jose, CA.

Levy, A.Y., Rajaraman, A., and Ordille, J.J. (1996a). Query-Answering Algorithms for Information Agents. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*. (pp. 40–47), Portland, OR.

Levy, A.Y., Rajaraman, A., and Ordille, J.J. (1996b). Querying Heterogeneous Information Sources Using Source Descriptions. In *Proceedings of the 22nd VLDB Conference*. (pp. 251–262), Bombay, India.

Levy, A.Y. and Rousset, M.-C. (1996a). CARIN: A Representation Language Integrating Rules and Description Logics. In *Proceedings of the European Conference on Artificial Intelligence*. (pp. 323–327), Budapest, Hungary.

Levy, A.Y. and Rousset, M.-C. (1996b). The Limits on Combining Recursive Horn Rules and Description Logics. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*. (pp. 557–584), Portland, OR.

Levy, A.Y., Srivastava, D., and Kirk, T. (1995b). Data Model and Query Evaluation in Global Information Systems. *Journal of Intelligent Information Systems, Special Issue on Networked Information Discovery and Retrieval*, 5(2), 121–143.

MacGregor, R. (1990). The Evolving Technology of Classification-Based Knowledge Representation Systems.

In J. Sowa (Ed.), *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan Kaufmann, pp. 385–400.

Nutt, W., Sagiv, Y., and Shurin, S. (1998). Deciding Equivalences Among Aggregate Queries. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. (pp. 214–223), Seattle, Washington.

Roth, M.T., Arya, M., Haas, L.M., Carey, M.J., Cody, W., Fagin, R., Schwarz, P.M., Thomas, J., and Wimmers, E.L. (1996). The Garlic Project. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 25(2), 557–558.

Roth, M.T. and Schwarz, P. (1997). Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. In *Proceedings of the Twenty-third International Conference on Very Large Data Bases*. (pp. 266–275), Athens, Greece.

Saraiya, Y. (1991). Subtree Elimination Algorithms in Deductive Databases. Ph.D. Thesis, Stanford University, Department of Computer Science.

Ullman, J.D. (1989). *Principles of Database and Knowledge-Base Systems*, Vol. 2. Rockville, Maryland: Computer Science Press.

Ullman, J.D. (1997). Information Integration Using Logical Views. In *Proceedings of the Sixth International Conference on Database Theory*. (pp. 19–40), Delphi, Greece.

Wiederhold, G. (1992). Mediators in the Architecture of Future Information Systems. *IEEE Computer*, pp. 38–49.

Jose Luis Ambite is a Research Associate at the USC Information Sciences Institute. He received his Ph.D. in Computer Science from the University of Southern California in 1998. His research interests include information integration, automated planning, databases, and knowledge representation.

Craig Knoblock is a Project Leader at the Information Sciences Institute and a Research Associate Professor in the Computer Science Department at the University of Southern California. He is also on the faculty of the Integrated Media Systems Center, which is a NSF Engineering Research Center at USC. He received his Ph.D. in Computer Science from Carnegie Mellon University in 1991 and joined USC that year. His current research interests include information agents, information integration, automated planning, and machine learning. He is one of the primary architects of the SIMS information mediator, which is a system for integrating heterogeneous data sources. He jointly leads the Ariadne and Theseus projects, which are addressing the problems of building agents for integrating and managing web-based information sources.

Ion Muslea is a Ph.D. Candidate in the Computer Science Department and the Information Sciences Institute at the University of Southern California. He received his B.S. at the Technical University of Cluj-Napoca (Romania), and his M.S. at West Virginia University. His research interests include machine learning, information integration, and information extraction.

Andrew G. Philpot is a research scientist at USC Information Sciences Institute. Prior to USC/ISI, he was at NASA Ames Research Center. He received his B.S.E. from Duke University in 1986 and his M.S from Stanford University in 1990. His research interests include planning and information integration.