Querying Schema Information

Vinay K. Chaudhri and Peter D. Karp
Artificial Intelligence Center
SRI International
333 Ravenswood Avenue, Menlo Park, CA, 94025, USA
Phone: (415)859-3368 Fax:(415)859-3735

Abstract

Schema queries can play an important role while retrieving information from multiple sources, for example, in query formulation and in query optimization. We identify four classes of schema queries that we have found useful while designing an application programming interface for frame representation systems (FRSs): taxonomic, frame structure, constraint and class comparison queries. We propose a scheme for direct support for these queries in a mediator language such as Object Query Language (OQL).

1 Introduction

A system that answers queries by accessing information residing in multiple sources has to perform the following tasks. Given a query, it has to determine which data items should be retrieved from which source, retrieve those data items, process the results, and present them to the user. Many of these tasks are performed by a software component called *Mediator* [Wid92].

Mediator usually issues queries to several data sources. The implementation of a mediator is considerably simplified if it is able to issue queries in one language instead of using a different language for each data source. Such an effect can be easily achieved by using one mediator language and providing wrappers for the data sources so that they can be accessed using the mediator language.

At a recent meeting of I*3/POB held at the University of Maryland at College Park, a subset of Object Query Language or OQL [Cat95] was proposed as a standard for Mediator language [BRU97]. OQL is a good initial proposal for a standard of a mediator language, but it provides inadequate facilities for querying schema information that can be extremely useful in mediation. Some existing information integration systems hand code the schema information in the mediator [CGMH+94] or manually construct an explicit model of the information content of a source [LRO96], but we believe that a mediator's ability to query a source for schema information and using it to determine the information content gives us more flexibility in adding new sources.

We have been working on Generic Frame Protocol (GFP), an application programming interface for object-oriented knowledge representation systems [KMG95, CFF+97]. We believe that GFP can be used as an inspiration for extending OQL to query schema information, which in turn can be used to build more powerful mediators.

GFP was originally motivated by a graphical browsing and editing application called GKB-EDITOR [PLK97]. We were interested in reusing GKB-Editor with multiple FRSs, and therefore, we built GFP wrappers for LOOM [Mac91], OCELOT, SIPE-2 [Wil90], THEO [MAC+89], and ONTOLINGUA [Gru93]. Our experience in using these wrappers in conjunction with GKB-Editor is described elsewhere [KCP97]. We anticipate that GFP will be adopted as an application programming interface in DARPA's high-performance KB initiative (see `http://www.teknowledge.com:80/HPKB/`) where its use will not be limited to just graphical browsing and editing applications.

Our goal in discussing the schema queries supported by GFP is not to propose GFP as a mediator language. Instead, we argue that the schema queries that have

Proceedings of the 4th KRDB Workshop
Athens, Greece, 30-August-1997

(F. Baader, M.A. Jeusfeld, W. Nutt, eds.)

http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-8/

been found useful in the context of GFP have a general utility and should be directly supported in the OQL which is being considered as a mediator language.

2 Using Schema Information in Answering Queries

We use a variation of an example published elsewhere [LRO96] in which the goal is to answer queries about used automobiles for sale. Suppose that we have three available information sources. Source 1 contains information about cars newer than 1990, and Source 2 about cars older than 1990. Source 3 is a car reviews database. Source 1 is a relational database. Source 2 is an object-oriented database (OODB) that defines three classes of cars – sports, economy, and vintage. Each of these classes has further subclasses, but we are not concerned with them at the moment. Source 2 records low blue book value and fair market value for the sports cars and economy cars and only fair market value for the vintage cars.

Schema information can be extremely useful to a user in formulating queries. For example, without any schema information, a user may formulate her query as Q1: Get the price and reviews of cars whose price is less than $15,000. But if the schema of Source 2 is made available, the user may decide that she is not really interested in vintage cars, and in addition to fair market price, blue book value is of interest. So, she can reformulate the query as Q2: Get the price, blue book value and reviews of cars whose price is less than $15,000 and which are not vintage cars. Q2 is likely to return a response much closer to the user's expectation. Thus, in some cases, knowing the class-subclass relationships and attributes of classes in a source can lead to better formulation of queries.

Schema information can also be useful in optimizing queries. Suppose the user poses a query Q2: Get the price and reviews of cars whose price is less than $10,000 and which are not older than 1992. The fact that Source 1 contains information about cars older than 1990 can be represented by a constraint that the maximum value of the year-of-manufacture attribute is 1990. The mediator then queries Source 1, determines that the values stored in it are outside the range of the query, and eliminates it from its access plan.

By itself, schema information is not sufficient for resolving semantic mismatches in the information residing in multiple sources [FDFP95]. For example, the *Price* attribute may represent different information in two sources — in one source it is blue book value, and in the other source, it is fair market value. We are not concerned with the problem of semantic mismatches in this paper.

3 Querying Schema Using GFP

GFP assumes a knowledge model of frame representation systems (FRSs). The basic representational units are *frames*, *slots*, and *facets* [CFF+97]. The frames are partitioned into classes and individuals. *Facets* represent properties of slots, for example, value type or cardinality constraints. In logical terms, class frames can be viewed as unary predicates, slots as binary predicates and facets as ternary predicates. A knowledge base (KB) is defined as a collection of frames and their associated slots, facets, and values.

In the terminology of OODBs, frames are like objects and slots are like attributes. OODBs do not have an analog of facets. From now on, we will use the terms *frames* and *objects*, and *slots* and *attributes* interchangeably.

GFP defines a collection of methods to query schema information. The GFP methods to query the schema information can be classified into four broad categories: taxonomic queries, frame structure queries, constraint queries, and class comparison queries. Table 2 lists a subset of GFP methods in each category.

The *taxonomic* queries allow us to query the class-subclass relationships. For example, `get-class-subclasses` allows us to determine all the subclasses of a class. The root classes can be determined using `get-kb-root-classes`.

The *frame structure* queries retrieve the slots and facets associated with a frame. For example, `get-frame-slots` returns all the slots associated with a frame. (A formal definition of what it means for a slot to be associated with a frame can be found elsewhere [CFF+97].)

The *constraint* queries allows us to query the facet information. As stated earlier, facets are used to represent constraints on slot values. Currently supported facet names are `:VALUE-TYPE`, `:CARDINALITY`, `:MINIMUM-CARDINALITY`, `:MAXIMUM-CARDINALITY`, `:INVERSE`, `:NUMERIC-MINIMUM`, `:NUMERIC-MAXIMUM`, `:SAME-VALUES`, `:SOME-VALUES`, `:NOT-SAME-VALUES`, `:SUBSET-OF-VALUES`, and `:COLLECTION-TYPE`. an informal definition of the facets is shown in Table 2. More formal definitions of facets are available elsewhere [CFF+97]. By using `get-facet-values` on the facets `:numeric-minimum` and `:numeric-maximum`, the range constraints on a slot can be obtained.

The *class comparison* queries support the inferences usually available only in description logic systems, such as LOOM [Mac91] and CLASSIC [BBMR89]. For example, `equivalent-p` examines the definitions of two classes and determines if they are equivalent in the sense that they have the same slots, slot values, facets, and facet values.

Table 1: Querying Schema Information by Using GFP

GFP Method	Brief Description
	Taxonomic Queries
`get-class-subclasses`	Returns a list of direct subclasses of a class
`get-class-superclasses`	Returns a list of direct superclasses of a class
`get-root-classes`	Returns a list of those classes that have no superclass
	Frame Structure Queries
`get-frame-slots`	Returns a list of all the slots of a frame
`get-frame-facets`	Returns a list of all the facets of a frame
	Constraint Queries
`get-facet-value`	Returns the value of a facet
`get-slot-facets`	Returns the list of facets applicable to a frame
	Class Comparison Queries
`equivalent-p`	Given classes class1 and class2, returns true when the extensions of class1 and class2 are the same
`consistent-classes-p`	Given classes class1 and class2, returns true if an instance could satisfy the definition of both classes simultaneously
`class-disjoint-p`	Given classes class1 and class2, returns true if they are incompatible, that is, an instance could not satisfy the definition of both classes simultaneously

Table 2: Facets Supported by GFP

Facet Name	Description
`:VALUE-TYPE`	A value C for facet `:VALUE-TYPE` of slot S of frame F means that every value of slot S of frame F must be an instance of the class C.
`:INVERSE`	A value S2 for facet `:INVERSE` of slot S1 of frame F means that if V is a value of S1 of F, then F is a value of S2 of V.
`:CARDINALITY`	A value N for facet `:CARDINALITY` on slot S on frame F means that slot S on frame F has N values.
`:MAXIMUM-CARDINALITY`	A value N for facet `MAXIMUM-CARDINALITY` of slot S of frame F means that slot S of frame F can have at most N values.
`:MINIMUM-CARDINALITY`	A value N for facet `MINIMUM-CARDINALITY` of slot S of frame F means that slot S of frame F has at least N values.
`:SAME-VALUES`	A value S2 for facet `:SAME-VALUES` of slot S1 of frame F, where S2 is a slot, means that the set of values of slot S1 of F is equal to the set of values of slot S2 of F.
`:NOT-SAME-VALUES`	A value S2 for facet `:NOT-SAME-VALUES` of slot S1 of frame F, where S2 is a slot, means that the set of values of slot S1 of F is not equal to the set of values of slot S2 of F.
`:SUBSET-OF-VALUES`	A value S2 for facet `:SUBSET-OF-VALUES` of slot S1 of frame F, where S2 is a slot, means that the set of values of slot S1 of F is a subset of the set of values of slot S2 of F.
`:NUMERIC-MINIMUM`	A value N for facet `:NUMERIC-MINIMUM` on slot S on frame F means that the minimum value of slot S on frame F is N.
`:NUMERIC-MAXIMUM`	A value N for facet `:NUMERIC-MAXIMUM` on slot S on frame F means that the minimum value of slot S on frame F is N.
`:SOME-VALUES`	A value V for own facet `:SOME-VALUES` of own slot S of frame F means that V is also a value of own slot S of F.
`:COLLECTION-TYPE`	The `:COLLECTION-TYPE` facet specifies whether multiple values of a slot are to be treated as a set, list, or bag.

4 Enhancing a Mediator Language to Query Schema

We believe that a mediator language should provide natural support for the four classes of queries considered in Section 3. We first briefly discuss the schema queries supported in the current relational DBMS products and then propose a scheme to enhance OQL to support schema queries.

The analog of a frame structure query such as `get-frame-slots` for a relational DBMS is to obtain a list of all the attributes of a relation. Traditionally, such queries have been answered by using the information in the data dictionary of a DBMS. The analog of a constraint query such as `get-facet-value` is to obtain the key of a relation or to obtain constraints attached to a relation. Even though constraint queries can be answered by querying the data dictionary, the current products do not offer the flexibility as we propose. For example, in ORACLE DBMS, it is possible to query the constraints associated with a table, but those constraints are returned as a string. Thus, if the numeric value of an attribute *Cost* was restricted to a positive integer, we will be returned the string "Cost $\geq$ 0". We then need to parse the result to determine that it represents a `:numeric-minimum` facet of GFP. The class comparison queries are outside the scope of relational DBMSs. Thus, RDBMSs support only a subset of schema queries that are useful for querying multiple sources.

Release 1.2 of OQL did not provide any support for taxonomic queries [Cat95]. To some extent, the problem will be rectified in the upcoming Release 2.0 of ODMG [Cat97], as the new data model includes a class *MetaObject* that has a relation called *DefiningScope* that will allow OQL to query the class-subclass relationships.

A possible way to incorporate taxonomic queries in OQL is to view each class relationship as a relation as proposed in XSQL [KKS92]. For example, the following XSQL query allows a variable to range over a class. XSQL syntax uses #X to distinguish the variables that range over classes.

```
SELECT #X WHERE Person subclassOf #X
```

The above query returns all the subclasses of the class `Person`. Each GFP method corresponding to taxonomic queries can be represented as a relation in an OQL query to provide a comprehensive support for querying the schema information. For example, `get-class-superclasses` can be represented by the relation superclassOf. Then the following query returns all superclasses of a class.

```
SELECT #X WHERE #X superclassOf Person
```

A similar technique can be used for supporting frame structure queries if we allow #X to represent a variable that ranges over attributes. For example, if we assume that the relation attributeOf represents the association of an attribute with a class, then the query

```
SELECT #X WHERE  #X attributeOf Person
```

returns all the attributes of the class Person. Alternative syntax for querying slots is possible [LSS96] and investigation of the relative syntax merits is left open for future research.

The Release 2.0 of the ODMG standard has limited facilities for querying the constraints on schema. It provides the method `getCardinality` to determine the cardinality of a relationship. We believe that more facilities should be provided to query constraints on an attribute or a relation. In GFP, the constraints are represented using facets. The ODMG data model does not support cardinality and range constraints, which is unfortunate because they can be extremely useful in optimizing queries in a heterogeneous database environment.

A possible approach to support queries on constraints is to define a method called `facet` that can be invoked during the OQL queries. For example, consider the class definition shown in Figure 4. Given the definition of `Person`, we show some sample queries and their expected results in Table 4.

The method `facet` can be system-generated when the schema is compiled. We believe that the ODMG data model should be extended to incorporate a larger set of constraints on the attribute and relationship values. The range and cardinality constraints supported by GFP are good initial candidates for inclusion in the ODMG data model.

To incorporate the class comparison queries in OQL, an approach similar to the one for other queries can be used. We can introduce a relation corresponding to each type of class comparison inference. For example, if consistentWith represents a relation that holds between two classes that are consistent, the query

```
SELECT #X WHERE Person consistentWith #X
```

returns all the classes that are consistent with the class `Person`.

5 Conclusions

In summary, we believe that the schema queries can be extremely useful in retrieving information from multiple sources. The example uses are support for query formulation and information for query optimization.

Figure 1: A Sample Class Definition

```
interface Person
(extent People)
{
        attribute String name;
        attribute Struct Address {Short number, String Street} address;
        relationship Person spouse inverse Person::spouse;
        attribute Integer age;
        relationship Set<Person> children inverse Person::parents;
        relationship List<Person> parents inverse children;
};
```

Table 3: Example Constraint Queries. The `NUMERIC-MINIMUM` facet cannot be determined using the schema of Figure 4.

Query	Expected Result
`select facet(p.name, value-type) from person p`	string
`select facet(p.spouse, value-type) from person p`	`Person`
`select facet(p.children, value-type) from person p`	`Set<Person>`
`select facet(p.children, collection-type) from person p`	`List`
`select facet(p.children, inverse) from person p`	`parents`
`select facet(p.age, numeric-minimum) from person p`	0

We identified four classes of schema queries that we have found useful while designing an application programming interface for FRSs: taxonomic, frame structure, constraint and class comparison queries. We believe that if direct support for the four classes of schema queries identified here is provided in OQL, it will be a more powerful mediator language.

Acknowledgments

This work was supported by Rome Laboratory contract F30602-96-C-0332. The contents of this article are solely the responsibility of the authors.

References

[BBMR89] Alexander Borgida, Ronald J. Brachman, Deborah L. McGuinness, and Lori Alperine Resnick. CLASSIC: A Structural Data Model for Objects. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pages 58–67, Portland, OR, 1989.

[BRU97] Peter Buneman, Louiqa Raschid, and Jeffrey Ullman. Mediator Languages — A Proposal for a Standard. *SIGMOD Record*, 26(1):39–44, 1997.

[Cat95] R. G. G. Cattell. *The Object Database Standard: ODMG-93, Release 1.2*. Morgan Kaufmann Publishers, Inc., 1995.

[Cat97] R. G. G. Cattell. *The Object Database Standard: ODMG-93, Release 2.0*. Morgan Kaufmann Publishers, Inc., 1997.

[CFF+97] Vinay K. Chaudhri, Adam Farquhar, Richard Fikes, Peter D. Karp, and James P. Rice. The Generic Frame Protocol 2.0. Technical report, Artificial Intelligence Center, SRI International, Menlo Park, CA, 21 July 1997. See `http://www.ai.sri.com/~gfp/spec.html`.

[CGMH+94] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS Project: Integration of Heterogeneous Information Sources. In *Proceedings of the 10th Meeting of the Information Processing Society of Japan*, pages 7–18, Tokyo, 1994.

[FDFP95] A. Farquhar, A. Dappert, R. Fikes, and W. Pratt. Integrating Information Sources Using Context Logic. In

AAAI-95 Spring Symposium on Information Gathering from Distributed Heterogeneous Environments, 1995.

[Gru93] T.R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199–220, 1993. URL for Ontolingua is `http://www-ksl.stanford.edu/knowledge-sharing/ontolingua/README.html`.

[KCP97] Peter D. Karp, Vinay K. Chaudhri, and Suzanne M. Paley. A Collaborative Environment for Authoring Large Knowledge Bases. Technical report, Artificial Intelligence Center, SRI International, Menlo Park, CA, submitted for publication, April 1997.

[KKS92] Michael Kifer, Won Kim, and Yehoshua Sagiv. Querying Object-Oriented Databases. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, pages 393–402, San Diego, May 1992.

[KMG95] P.D. Karp, K. Myers, and T. Gruber. The Generic Frame Protocol. In *Proceedings of the 1995 International Joint Conference on Artificial Intelligence*, pages 768–774, 1995. See also WWW URL `ftp://ftp.ai.sri.com/pub/papers/karp-gfp95.ps.Z`.

[LRO96] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *Proceedings of the 22nd International Conference on Very Large Databases*, Bombay, 1996.

[LSS96] Laks V. S. Lakshmanan, Fereidoon Sadri, and Iyer N. Subramanian. SchemaSQL — A Language for Interoperability in Relational Multi-database Systems. In *Proceedings of the 22nd International Conference on Very Large Databases*, Bombay, 1996.

[MAC+89] T.M. Mitchell, J. Allen, P. Chalasani, J. Cheng, E. Etzioni, M. Ringuette, and J.C. Schlimmer. THEO: A Framework for Self-Improving Systems. In *Architectures for Intelligence*. Erlbaum, 1989.

[Mac91] R. MacGregor. The Evolving Technology of Classification-based Knowledge Representation Systems. In J. Sowa, editor, *Principles of Semantic Networks*, pages 385–400. Morgan Kaufmann Publishers, Los Altos, CA, 1991.

[PLK97] Suzanne M. Paley, John D. Lowrance, and Peter D. Karp. A Generic Knowledge Base Browser and Editor. In *Proceedings of the Ninth Conference on Innovative Applications of Artificial Intelligence*, 1997.

[Wid92] Gio Widerhold. Mediators in the Architectures of Future Information Systems. *IEEE Computer*, 25:38–49, 1992.

[Wil90] D.E. Wilkins. Can AI planners solve practical problems? *Computational Intelligence*, 6(4):232–246, November 1990.