Distributed Information Mediation and Query Processing in a CORBA Environment *

Stanley Y. W. Su and Tsae-Feng Yu
Database Systems Research and Development Center
Department of Computer and Information Science and Engineering
University of Florida
{su,yu}@cise.ufl.edu

Abstract

A heterogeneous information system can be built on a large number of component systems which are interconnected by a LAN or WAN. The data stored in these component systems are likely to have very different naming, structural and semantic representations. The querying facility of the information system needs to couple with an information mediation facility to resolve data heterogeneity problems so that a user can issue queries in the terms and receive data in the data representations that are familiar to him/her. In this work, we introduce an object-oriented modeling language for modeling the data resources and object services of the component systems, and a mediation specification language for explicitly specifying the similarities and differences among data representations, as well as the methods needed to do data conversions. Based on the schemas and the mediation specifications defined for the component systems, a compiler has been developed to generate enhanced program bindings for the component systems. These bindings contain subquery, rule and mediation processing code to perform distributed mediation, query and active rule processing tasks at run-time over the OMG's CORBA communication infrastructure. System efficiency is thus achieved by the compilation approach and the distributed processing of generated code.

1 Introduction

In a heterogeneous information system, component systems operating on different computing platforms need to exchange and share data resources. Data residing in these systems would generally have different structural and semantic representations. A traditional approach taken by the existing heterogeneous systems is to establish an "integrated global schema" over the conceptual models of the data resources stored in these systems. The integrated global schema forces all semantic ambiguities, naming conflicts, and structural/semantic discrepancies among dissimilar systems to be resolved at the time when the global schema is being designed. This means that some users of

*This research is supported by the Advanced Research Project Agency under ARPA Order No. B761-00 and managed by the United States Air Force under contract F33615-94-2-4447. This is a part of the R&D effort of the NIIIP Consortium. The views and conclusions contained in this paper are those of the authors and should not be interpreted necessarily representing the official policies, either expressed or implied, of all the NIIIP Consortium members, the Advanced Research Projects Agency, or the United States Government.

the existing systems will be forced to view and make reference to the global database in a way not familiar to them, both structurally and semantically. More recent thinking in the database community is to "mediate" dissimilar data representations at run-time instead of "integrating" them at build-time [CHE88, WIE92]. The term "mediation" has been defined in a very broad sense in [WIE94]. Almost all tasks which facilitate the communication and interchange of dissimilar data or the resolution of disagreements or disputes can be considered as some form of mediation tasks. However, as pointed out in the literature [BRE90, CHAL94, CHA91, GOH94, HAM93, KIM91, VEN91], the most critical problem in a heterogeneous information system is the resolution of various kinds of data heterogeneities. This problem has not been effectively solved, especially in a large-scale system environment. Therefore, in this work, we restrict the domain of mediation to information mediation which deals with problems of naming, structural and semantic differences.

The basic idea of information mediation is to allow a user of a heterogeneous information system to view data the way he/she wants and is accustomed to. The heterogeneous information system will transform the semantic and/or structural representations of data at run-time so that the data retrieved from the dissimilar component systems can be converted, assembled and presented to the user in the structure and semantics familiar to him/her.

Information mediation needs to be closely-coupled with distributed query processing. A global query is issued by a client to a Distributed Query Processor (DQP) which decomposes the global query into a number of subqueries depending on the locations of the data referenced in the query. Each subquery is to be processed against some data residing in a different component system. Since the global query uses terms (object class names, attribute names, data value representations, etc) which are familiar to the user (i.e., based on the data representation defined in a component schema), the terms in a subquery may have to be converted by a mediator to fit the naming convention and the syntactic and semantic representations of another component system (i.e., a server) before the subquery can be processed by it. Each server would then translate the mediated subquery into a native query, command or application interface (API) processable by the server, and the retrieved data will have to be transformed into a standard data interchange form before they are returned to the mediator. The returned data from the servers of all the subqueries may have to be converted by the mediator to conform to the user's view of data before they can be correctly assembled by a Data Assembler (a component of DQP). The assembled data are then forwarded to the client.

To carry out the mediation tasks in the above scenario, the approach taken in this work is

to use a high-level object-oriented modeling language (NCL [SU96]) to uniformly model the data resources as well as the component systems as object classes, each of which is defined in terms of attributes, associations, methods and event-condition-action-alternativeaction rules. Multiple component schemas having dissimilar data representations are produced in the modeling process. A high-level mediation specification language is then used to specify the mediation specification that captures the naming, structural, and semantic similarities and differences among component schemas. The mediation specification is then compiled to generate a set of object classes which model the DQP, the subquery processors and mediators. These classes contain active mediation rules for triggering the mediation operations at run-time. The specifications of the component systems and the generated classes are combined to form a mediated global schema which is used by the user to issue queries and by a compiler to generate executable code to support distributed mediation and query processing.

The main differences between our work and some existing mediation efforts [AMB94, CHAW94, GOH94, FLO96, SAU96] are: 1) the use of an object-oriented modeling language, which combines the features of two standard languages (ISO's EXPRESS and OMG's IDL) and the association and rule specification facilities of our own modeling language K [SHY96], for defining the information resources, as well as the component systems of a heterogeneous network, 2) the use of a high-level mediation specification language (instead of low-level mediation logic rules) for mediation specifications, 3) the use of a compilation approach (instead of an interpretive approach) to automatically generate distributed mediation and query processing code so that the overloading problem of a centralized mediation system can be avoided, and 4) the use of a set of mediated component schemas (instead of a single or multiple integrated schemas) to allow the user to issue queries based on the schema familiar to him/her and receive data in his/her own view. A prototype distributed mediation and query processing system with the above features has been implemented as a part of a DARPA-supported project entitled "National Industrial Information Infrastructure Protocols (NIIIP)".

The remainder of this paper is organized as follows. Section 2 defines and categorizes data heterogeneity problems. Section 3 presents our approach to dealing with these problems. It describes the object-oriented information modeling language, the mediation specification language with some mediation specification examples, the translation and code generation process, and the distributed query processing and mediation. Section 4 summarizes the main features of this proposed approach and reports on the implementation status.

2 Problems of Data Heterogeneity

Data heterogeneity problems have been thoroughly discussed in several publications [BRE90, CHA91, KIM91, SU91, VEN91, HAM93, CHAL94, GOH94]. Based on the work of Goh, Madnick and Siegel [GOH94], these problems are categorized as schematic heterogeneity and semantic heterogeneity.

Schematic heterogeneity includes two types of problems: naming and structural conflicts. The naming conflicts include the synonym and homonym problems on both attribute and entity type names. The structural conflicts are due to different ways of modeling the same piece of information. For example, in Figure 1, the company name, "IBM", can be used as an entity type name, an attribute name, and a value of an attribute in different systems.

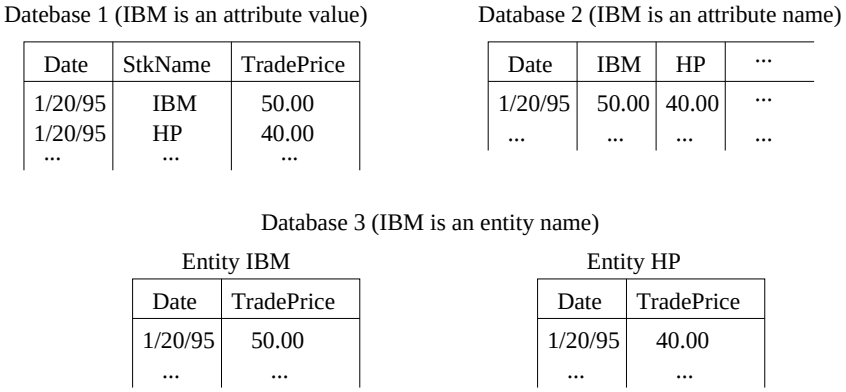

Figure 1: Three Examples of Showing Structural Conflicts

Semantic heterogeneity is due to different representations of data values. It includes naming and other representational conflicts. The naming conflicts in attribute values are seen as synonyms (e.g., 'IBM' and 'IBM Corp') and homonyms (e.g., persons with the same name). Other representational conflicts in this category include: (1) measurement conflicts (US Dollar vs. Yen), (2) representation conflicts (New York Stock Exchange representation vs. decimal representation), (3) confounding conflicts (e.g., latest closing price vs. latest trade price), (4) granularity conflicts (e.g., monthly pay vs. yearly pay), and (5) domain type conflicts (e.g., numerical type vs. string type).

3 Approach

3.1 Object-oriented Modeling

A heterogeneous information system may contain component systems that run on different computing platforms and use different types of data management systems. CORBA's approach to achieve data and program sharing is to model all data and software resources as distributed objects, and

their interfaces are uniformly defined in an Interface Definition Language (IDL). However, IDL has a very limited number of modeling constructs. Much of the semantic information associated with data entities and software systems can be lost in the IDL specifications. In our work, a semantically-rich modeling language is used to model the resources of the component systems resulting in a number of component schemas. Each component schema preserves the naming, structural and semantic representations of the data stored in each component system. For details of the modeling language, the readers are referred to [SU96].

3.2 Mediation Specification

A high-level mediation language is introduced to explicitly specify the similarities and differences among data elements and the mediation operations needed for modifying queries and converting data from one representation to another. For a number of component schemas that contain semantically related data, a mediation specification is defined by the mediation system administrator.

The design of the mediation specification language meets the following requirements: (1) the language should be able to explicitly specify the naming, structural and semantic relationship of heterogeneous component systems, (2) the syntax of the language should be high-level and easy to use, and (3) the syntactic constructs should conform to the common modeling language (i.e., NCL) so that they can be easily translated into NCL class specifications.

The heterogeneity problems discussed in Section 2 provide a good guideline in our design of the mediation language. The overall structure of a mediation specification is shown below:

```
SCHEMA Med_Schema;
  USE FROM schema_1(entity_1,...);
  USE FROM schema_2(entity_2,...);

  ENTITY super_entity_id
    ABSTRACT SUPERTYPE OF (supertype_expression);
    ENTITY EQUIVALENCE (sch_1::entity_1,sch_2::entity_2,...) ;
    ATTRIBUTE EQUIVALENCE [(sch_1::entity_1.attr_1,sch_2::en tity_2.attr_2,...);
                                        | (attr_set_1, attr_set_2,...);]
      [VALUE EQUIVALENCE((sch_1::entity_1.attr_1,conv_method (sch_2::entity_2.attr_2),...);
                                    | (sch_1::entity_1.attr_1,conv_method(attr_set_2),...);
                                  ); ]
      [WHERE
            simple_condition;
            ... ]
      ...
  END_ENTITY;
```

...
END_SCHEMA;

The mediation language conforms to the standard information specification language, EXPRESS [ISO92] (a part of NCL) by using some of its keywords. Four additional syntactic constructs are introduced in the mediation language. They are:

- The ENTITY EQUIVALENCE clause is used to declare the equivalence relationship between two or more entity types and to resolve the problem of synonymous entity type names. Entity type names enclosed in the clause are declared to be synonyms and are semantically equivalent. The synonym relationship is used to generate code to do query modifications (a method of the Mediator) by the mediation language compiler.

- The ATTRIBUTE EQUIVALENCE clause is used to represent the synonym relationship among a set of attributes, each of which can be composite. These attributes are of the same meaning and their values are convertible. The clause is also used to generate part of the implementation code for query modification. The information on attribute name mappings is embedded in the code of that method.

- The VALUE EQUIVALENCE clause specifies the method to be used to perform data conversions between two different systems. The data conversion method specified in the clause is to convert data of one or a set of attributes into the representation of another attribute. Data conversions are needed to resolve the semantic heterogeneity problems, which may involve both irregular (e.g., synonymous data values) and regular (e.g., unit or measurement conflict) data mappings. The synonym problem in data values can be resolved by pairwise mappings of equivalent data values (e.g., the value 'IBM' in system A is equivalent to the value 'IBM Corp' in system B) in the implementation of the conversion method. Similarly, mathematical functions (usually simple ones) can be embedded in the conversion method to deal with regular data mappings. The implementations of conversion methods can be done in NCL or other programming languages.

- The WHERE clause following the ATTRIBUTE EQUIVALENCE clause is used to resolve both the homonym problem on values and the structural conflict between two systems. Homonymous values are identified by specifying the equality conditions of key attributes in the WHERE clause. The attribute values are identical only when the key attribute values

are the same. In data modeling, the same piece of information can be modeled in different ways, which causes the problem of structural conflicts. The conversion between two attribute values is possible when a special condition is true (e.g., an entity name is equal to a particular attribute value). It represents a conditional mapping relationship between related classes and is needed in query modification. The WHERE clause is also used to specify the special condition for the mediation.

It is noted that, by default, all entity and attribute names defined in different systems are assumed to be unrelated, unless ENTITY EQUIVALENCE and ATTRIBUTE EQUIVALENCE clauses are used to explicitly specify their synonymous relationships. The homonym problem on attribute values is handled by using the WHERE clause to explicitly specify the equality condition of key attributes as explained above.

The following examples illustrate the use of the mediation language to handle the heterogeneity problems discussed in this paper.

(1) Example of Schematic Heterogeneity: Naming and Structural Conflicts

In Figure 2, stock information is modeled in different ways in schemas DB_1 and DB_2. For example, "IBM" is an attribute value of StkCode in DB_1, but is an attribute name in DB_2. The value of TradePrice in DB_1 is equal to that of IBM in DB_2 only if StkCode = "IBM" in DB_1. The similar condition needs to hold for HP stock price in the two semantically related component schemas. Also, Date and Stkcode of DB_1 form a composite key, whereas Date of DB_2 is the key (indicated by a double-slash link). Stock_1_2 is the generalization of the two Stock classes in the two component schemas with an assumed constraint of SI (i.e., Set-Intersection which is the same as the ANDOR constraint in EXPRESS).

```
SCHEMA Mediation;
  USE FROM DB_1 (Stock);
  USE FROM DB_2 (Stock);

  ENTITY Stock_1_2;
    ABSTRACT SUPERTYPE OF (DB_1::Stock ANDOR DB_2::Stock);
    ENTITY EQUIVALENCE (DB_1::Stock, DB_2::Stock);
      ATTRIBUTE EQUIVALENCE (DB_1::Stock.Date, DB_2::Stock.Date);
      ATTRIBUTE EQUIVALENCE (DB_1::Stock.TradePrice, DB_2::Stock.HP);
        WHERE
          DB_1::Stock.StkCode = "HP";
      (* DB_1::Stock.TradePrice and DB_2::Stock.HP are equivalent only
         when DB_1::Stock.StkCode is equal to "HP" *)
      ATTRIBUTE EQUIVALENCE (DB_1::Stock.TradePrice, DB_2::Stock.IBM);
        WHERE
          DB_1::Stock.StkCode = "IBM";
      (* DB_1::Stock.TradePrice and DB_2::Stock.IBM are equivalent only
         when DB_1::Stock.StkCode is equal to "IBM" *)
```

```
  END_ENTITY;
END_SCHEMA;
```

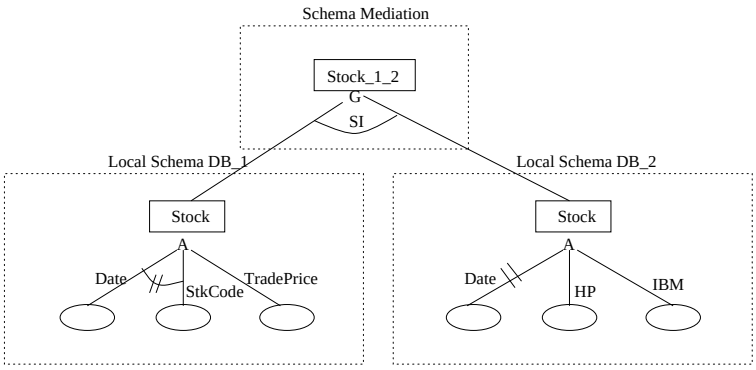

Figure 2: Example of Structural Conflict

(2) Example of Semantic Heterogeneity: Measurement Conflicts

The example shown in Figure 3 represents a group of heterogeneity problems in the category of semantic heterogeneity which are resolvable by using conversion methods. In this example, two component systems USA and Japan contain stock data which are represented in different currencies (i.e., US Dollar and Yen). The two currencies are convertible by applying some simple conversion functions (e.g., USD_to_Yen and Yen_to_USD). For some other semantic heterogeneities, including representation, confounding, granularity and data type conflicts, they can be resolved by data conversion methods specified in mediation specifications. Their program code needs to be provided by the mediation system administrator.

```
SCHEMA Mediation;
  USE FROM Japan (Stock);
  USE FROM US (Stock);

  ENTITY Stock_JP_US;
    ABSTRACT SUPERTYPE OF (Japan::Stock ANDOR US::Stock);
    ENTITY EQUIVALENCE (Japan::Stock, US::Stock);
      ATTRIBUTE EQUIVALENCE (Japan::Stock.Price, US::Stock.Price);
        VALUE EQUIVALENCE (
          (Japan::Stock.Price, USD_to_Yen(US::Stock.Price));
          (US::Stock.Price, Yen_to_USD(Japan::Stock.Price));
        );
        (* Data conversions between two currencies *)
  END_ENTITY;

END_SCHEMA;
```

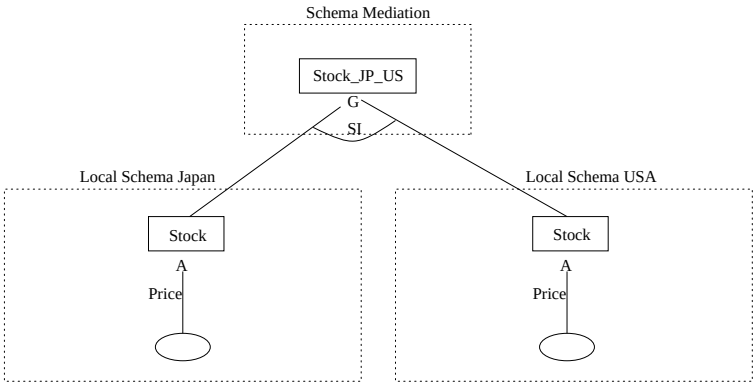

Figure 3: Example of Semantic Heterogeneity: Measurement Conflicts

3.3 Translation of Mediation Specifications

The mediation language illustrated by examples in the last section is used to mediate a number of component schemas defined in NCL which capture the semantics of some existing systems. Given some mediation specifications, a mediation language translator is used to generate the "mediation elements" defined in NCL based on these specifications. This set of mediation elements contains the following object classes with active mediation rules and methods for carrying out the run-time mediation tasks.

- *Supertype entity classes*: These superclasses generalize the ENTITY classes of different component systems if they contain semantically related objects. They upward inherit the attributes and methods associated with these entity classes. The specifications of these superclasses also capture the set membership constraints among the related ENTITY classes, such as Set-Equality, Set-Exclusion, or Set-Intersection. This information is useful for query optimization (e.g., if two component systems contain identical objects and their data are the same, then there is no need to send a query to both systems).

- *Mediator object classes*: These classes model the mediators which are generated based on the mediation specifications associated with semantically related schemas. For achieving processing efficiency, the Mediator classes are distributed and linked with different component systems at different sites so that mediation operations can be processed locally. Each Mediator contains methods which perform query modification and data conversion. These methods are invoked by the Subquery Processor of the component system when mediation is needed.

- *Distributed Query Processor (DQP) object class*: The Distributed Query Processor class contains a method which performs distributed query processing in the heterogeneous system. This method with a query as its parameter is invoked by the user/application program. The implementation of this method is based on a built-in query processing algorithm. The DQP class has an ECAA mediation rule which, when triggered, calls a knowledgebase management system (OSAM*.KBMS [SU95]) to obtain the meta data for locating the information sources and propagating subqueries.

```
Algorithm of the DQP's global query execution

(* Input: a textual OQL global query *)
(* Output: a set of data results *)

1. Query Parsing: Parse the global query into a parse tree structure
2. Tree Transformation: Transform the tree to generate simple subqueries,
   each of which involves only a single class.
3. Subquery Propagation: For each subquery, a mediation rule is triggered
   to look up the meta data from the KBMS to locate the information sources
   which can answer the subquery. For different information sources, more
   subqueries are propagated.
4. Subquery Dispatching: Dispatch each subquery to the information source
   for processing.
5. Global OID Assignment: Data returned from information sources are already
   mediated and uniformaly represented by the local mediator before received
   by the DQP. The DQP first assigns global object IDs for the data instances
   based on the key attribute values. Then, duplicated data instances are
   identified.
6. Data Assembly: First, perform union operations on the data generated from
   the propagated subqueries (i.e., generating a set of data results for simple
   subqueries).  Then, perform join operations to combine the data results
   based on the join condition(s) to produce the final result.
7. Data returning: Return the final result to the client.
```

- *Subquery Processor (SQP) object classes*: These classes are Subquery Processors generated for the component systems. Each SQP receives a subquery from the DQP and calls the Mediator, if a subquery conversion is needed, to generate a mediated subquery. The SQP then sends the mediated subquery to the wrapper which converts it into a native query, command or API processable by the component system. The SQP object class contains a main method which triggers two mediation rules. One is to modify the subquery to conform to the naming and structural convention of the component system before the main method is executed. The other is to convert the returned data from its local representation to the one expected by the user after the main method finishes its execution.

- *Mediation methods*: Based on the mediation clauses (i.e., ENTITY EQUIVALENCE, ATTRIBUTE EQUIVALENCE, VALUE EQUIVALENCE and WHERE), the implementation code for the mediation methods, including query modification, data conversion, etc., can

either be generated automatically or provided by the system administrator.

The generated mediation elements and the component schemas defined in NCL form a mediated global schema which is then compiled and stored in the KBMS. The meta information of the mediated global schema is accessed by the DQP at run-time to locate component systems that contain the relevant data.

The following example illustrates the mediation language translation process.

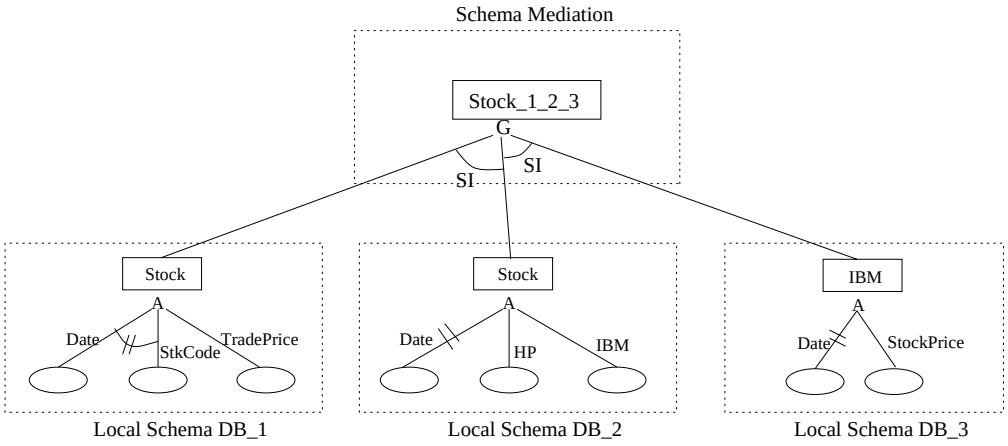

Figure 4: Mediation Schema on Top of Three Component Schemas

```
SCHEMA Mediation;
  USE FROM DB_1(Stock);
  USE FROM DB_2(Stock);
  USE FROM DB_3(IBM);

  ENTITY Stock_1_2_3
    ABSTRACT SUPERTYPE OF (DB_1::Stock ANDOR DB_2::Stock ANDOR DB_3::IBM);
    ENTITY EQUIVALENCE (DB_1::Stock,DB_2::Stock,DB_3::IBM);
      ATTRIBUTE EQUIVALENCE (DB_1::Stock.Date,DB_2::Stock.Date,DB_3::IBM.Date);
      ATTRIBUTE EQUIVALENCE (DB_1::Stock.TradePrice,DB_2::Stock.HP);
        VALUE EQUIVALENCE(
          (DB_1::Stock.TradePrice,Decimal_to_NewYork(DB_2::Stock.HP));
          (DB_2::Stock.HP,NewYork_to_Decimal(DB_1::Stock.TradePrice)));
        WHERE
              DB_1::Stock.StkCode = 'HP';
        (* DB_1::Stock.TradePrice and DB_2::Stock.HP are equivalent only when
           DB_1::Stock.StkCode is equal to 'HP' *)
      ATTRIBUTE EQUIVALENCE (DB_1::Stock.TradePrice,DB_2::Stock.IBM,
                                                DB_3::IBM.StockPrice);
        VALUE EQUIVALENCE(
          (DB_1::Stock.TradePrice,Decimal_to_NewYork(DB_2::Stock.IBM));
          (DB_2::Stock.IBM,NewYork_to_Decimal(DB_1::Stock.TradePrice));
```

```
    (DB_1::Stock.TradePrice,Decimal_to_NewYork(DB_3::IBM.StockPrice));
    (DB_3::IBM.StockPrice,NewYork_to_Decimal(DB_1::Stock.TradePrice)));
  WHERE
      DB_1::Stock.StkCode = 'IBM';
  (* DB_1::Stock.TradePrice is equivalent to DB_2::Stock.IBM and DB_3::IBM.StockPrice
    only when DB_1::Stock.StkCode is equal to 'IBM' *)
END_ENTITY;

END_SCHEMA;
```

Figure 4 shows that three component databases, DB_1, DB_2, and DB_3, which contain semantically related stock information, are mediated by the mediation specification shown above. In Figure 4, Stock_1_2_3 is a generalization (or superclass) of DB_1::Stock, DB_2::Stock and DB_3::IBM with two assumed pair-wise constraints of Set-Intersection (SI). The constraint specifies that objects in each pair of subclasses can overlap. The main difference among these three databases is that the company name is modeled as an attribute value, an attribute name, and an entity name in DB_1, DB_2 and DB_3, respectively. Additionally, the data representation of the stock price in DB_1 is different from that in DB_2 and DB_3. In DB_1, the stock price is represented in the New York Stock Exchange representation, (e.g., 6\08); whereas, in DB_2 and DB_3, it is in the common decimal representation (e.g., 6.5).

Suppose a client wants to get the stock price of "IBM" by issuing a global query against the DQP based on the view of schema DB_1, as shown in Figure 5. The global query service is a method of the DQP that contains the program code of the query execution plan. The DQP first parses, checks and decomposes the global query into simple subqueries. Since the query given above is already simple (i.e., referencing a single class), it is not decomposed. The query is stored in a data structure for initialization. After the query is initialized, a mediation rule (i.e., rule_query_propagation shown in the translated results given below) associated with the DQP is triggered to access the meta-information from the KBMS. The meta-information of the mediated global schema indicates that two other information sources (i.e., DB_2 and DB_3) contain related data. Thus, three subquery instances are established by "propagating" the global query. Since the global query is issued based on schema DB_1, the subqueries generated for DB_2 and DB_3 need to be modified to conform to the naming convention of these two systems. The mediation rules associated with the SQP of the component system are triggered to invoke the methods of the mediator to 1) modify the subquery properly before the mediated subquery is executed in the local server and 2) convert data into the representation expected by the client (i.e., the New York Stock Exchange representation) after the

mediated subquery result is returned. Once data are converted by the mediator into a uniform representation, the global query processor assembles the data and returns the assembled data to the client.

The following object classes defined in NCL is part of the translated results for the mediation specification given above.

```
(* Common data structures of Query and returned data *)
(* object class 'Query' for storing query statement and data result *)
DEFINE ENTITY Query;
    query_node: Query_node;      (* query node *)
    s_schema: STRING;            (* source schema name *)
    t_schema: STRING;            (* target schema name *)
    result: SET OF Data;         (* query result *)
END_DEFINE;

(* Object class 'Data' is defined for standard data interchange *)
DEFINE ENTITY Data;
   value_set: SET OF Generic_Data_Type;(* result values *)
   size: INTEGER;                      (* size of SET *)
   data_type: STRING;                  (* data type *)
   s_attr: STRING;                     (* attribute name in source schema *)
   t_attr: STRING;                     (* attribute name in target schema *)
END_DEFINE;

(* Generic Data Type:a union of all possible data types *)
DEFINE TYPE Generic_Data_Type=
            SELECT OF (string_type, number_type, boolean_type, Data);
END_DEFINE;

(* Mediator class in DB_1 generated from 'Stock' mediation specification *)
DEFINE ENTITY Mediator IN DB_1;
METHODS:
 (* Two main methods: modify_query, data_conversion *)
 METHOD modify_query(INOUT ps:Query):VOID;
 METHOD data_conversion (INOUT result:SET OF Data):VOID;
 METHOD convert(INOUT data:Data):VOID;
 ...
END_DEFINE;

(* Implementation of Method modify_query() *)
METHOD Mediator::modify_query(INOUT ps:Query):VOID;
  IF (ps.s_schema= 'DB_2' AND ps.t_schema= 'DB_1')
  THEN
   change_name_add_cond('IBM','TradePrice',ps, 'StkCode="IBM"');
   ...
  END_IF; ...
END_METHOD;

(* Implementation of Method data_conversion() *)
METHOD Mediator::data_conversion(INOUT result:SET OF Data) :VOID;
  (* For each column of data *)
    IF (result[i].s_attr <> result[i].t_attr)
    THEN convert (result[i]);
    END_IF;
END_METHOD;

(* Subrountine of data_conversion() *)
METHOD Mediator::convert(INOUT data:Data):VOID;
  (* For each data value in the column *)
    IF (data.s_attr = 'DB_1.Stock.TradePrice' AND data.t_attr = 'DB_2.Stock.HP')
    THEN NewYork_to_Decimal(data.value_set[i]);
    END_IF; ...
END_METHOD;

(* Definition of class 'Distributed Query_Processor' *)
DEFINE ENTITY DQP IN MEDIATION;
  global_query: Query;      (* for global query *)
  subqueries: SET OF Query; (* for subqueries *)
```

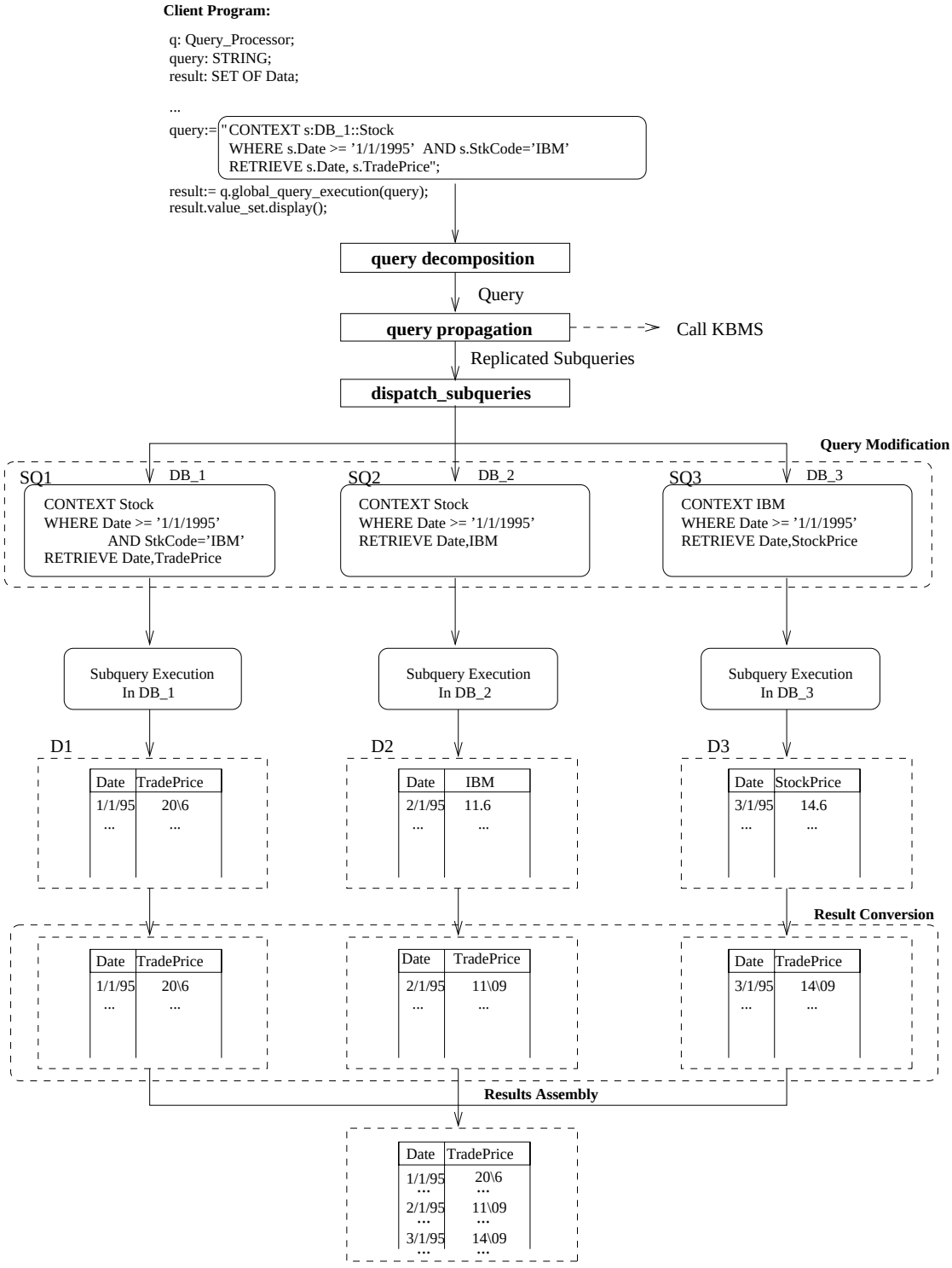

Figure 5: Example of a Global Query Processing

```
 METHODS:
  (* global_query_execution() is the main method *)
  METHOD global_query_execution(IN query_txt:STRING) :SET OF Data;
  ...
 RULES:
  (* rule triggered to get the meta data from the KBMS to locate information
     sources *)
  RULE rule_query_propagation;
  TRIGGER IMMEDIATE AFTER subq_initialization(s)
  ACTION
    prop_subqs:= KBMS::propagate_queries(s);
  END_RULE;
END_DEFINE;

METHOD DQP::global_query_execution (IN query_txt: STRING): SET OF Data;
  ...
  query_initialization(query_txt);
  IF (syntax_semantic_checking() == FALSE)
  THEN
    error_handler();
  END_IF;
  subquery_generation(global_query.Query_node);

  FOREACH (s:subqueries)
    subq_initialization(s);
    FOREACH (ps:prop_subqs)
      dispatch_subquery(ps);
    END_FOREACH;
    assemble_mediated_results(s,prop_subqs);
  END_FOREACH;
  result_merge_join();
  RETURN (global_query.result);
END_METHOD;

(* Component System DB_1 *)
DEFINE SCHEMA DB_1;
END_DEFINE;

DEFINE ENTITY SQP IN DB_1;
  m: Mediator;
METHODS:
  METHOD query_execution(INOUT q_obj: Query_obj): VOID;
RULES:
  (* rule triggered to modify the subquery into the view of local systems *)
  RULE rule_query_modification;
  TRIGGER BEFORE query_execution(q_obj)
  CONDITION q_obj.s_schema <> q_obj.t_schema
  ACTION
    m.modify_query(q_obj);
  END_RULE;

  (* rule triggered to convert the data into the view of the original query *)
  RULE rule_data_conversion;
  TRIGGER IMMEDIATE AFTER query_execution(q_obj)
  ACTION
    m.data_conversion(q_obj.result);
  END_RULE;
END_DEFINE;
```

3.4 Generation of CORBA Enhanced Program Binding

The above NCL class specifications are used to generate program code for the DQP, SQPs and mediators. The code for SQPs and Mediators are linked with the program code of component systems. As shown in Figure 6, component systems, DQP, client and KBMS communicate with each other over ORB which is the CORBA communication infrastructure. CORBA is a client/server architecture in which the client makes a service request by issuing a method call which is trans-

parently dispatched to the server. In the architecture, each component system can be a client, a server or both.

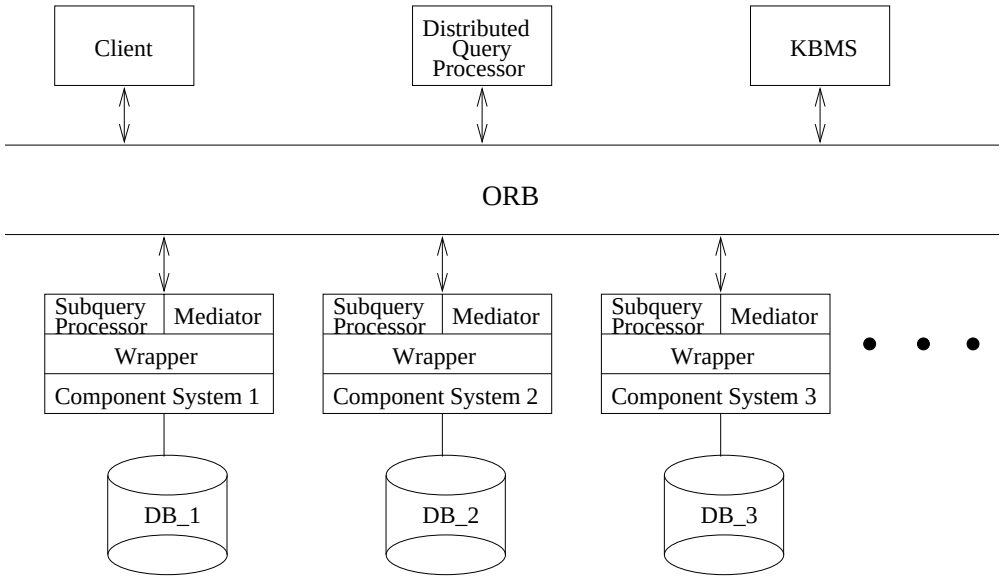

Figure 6: CORBA-based Mediation System Architecture

To carry out the code generation for use in the CORBA environment, an NCL compiler is developed to translate the NCL specifications into CORBA program bindings. All the semantic properties captured by NCL, but not IDL (e.g., keyword constraints and associations), are first converted into ECAA rules. These and other explicitly specified ECAA rules are translated into C or C++ code. Parts of NCL class definitions which are equivalent to IDL specifications are translated into IDL specifications. The IDL specifications are then translated by the IDL compiler to produce C or C++ stubs for the clients and C or C++ skeletons for the servers. The rule code, mediation code, and subquery processing code are incorporated in the skeletons to form the so-called enhanced bindings for the servers, as illustrated in the top diagram of Figure 7. We shall use the ECAA rules in the generated SQP class, which is shown at the end of Section 3.3, as an example to illustrate the process of generating enhanced program bindings.

Figure 7 shows the compilation of a method (query_execution) with its associated before- and immediate-after rules (rule_query_modification and rule_data_conversion) into the C or C++ code. During the compilation of the SQP class, the NCL compiler translates each rule into a C or C++ method. For the rule named rule_query_modification, the C or C++ code in the method named query_execution_R1 will check the condition ps.s_schema <> ps.t_schema and call the method modify_query if the condition evaluates to True. Similarly, for the rule named rule_data_conversion,

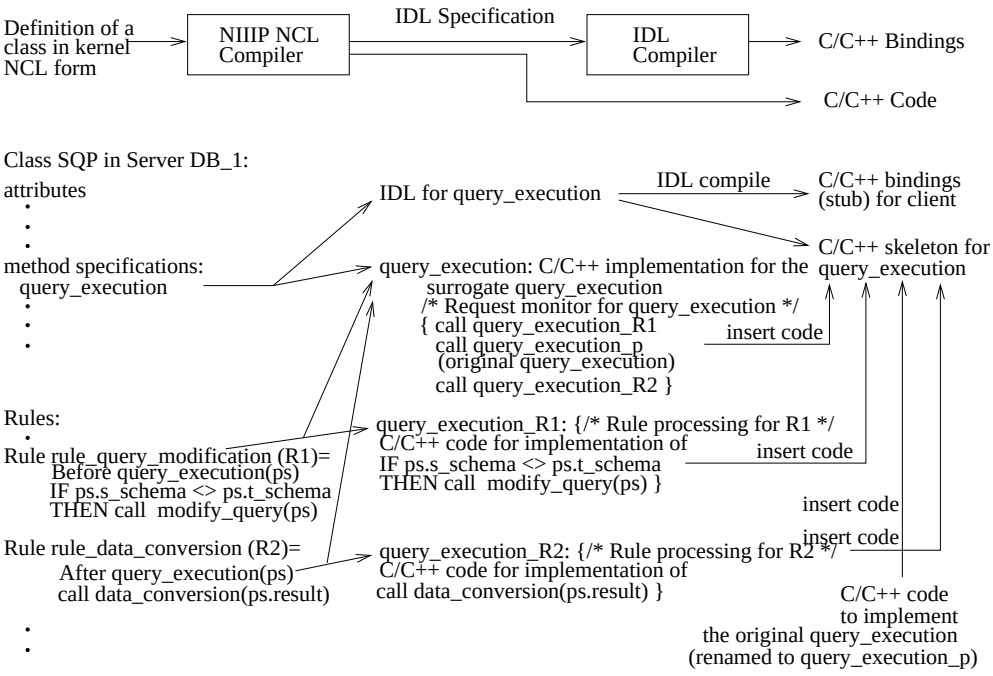

Figure 7: Compilation of an NCL Method query_execution() and Its Associated Rules

a C or C++ method named query_execution_R2 is generated.

For each method in the class, an equivalent IDL specification is generated. For example, an IDL specification is generated for the method query_execution and is renamed as query_execution_p. Furthermore, a new method named query_execution (i.e., a surrogate query_execution in C or C++ code) is generated. The new implementation consists of three method calls. First, a call to method query_execution_R1 is made to process the before-rule rule_query_modification. Then, a call is made to the original implementation of the method query_execution, which has been renamed as query_execution_p, to perform the subquery processing. Finally, a call is made to method query_execution_R2 to process the immediate-after-rule rule_data_conversion.

In the final step of the compilation process, the IDL compiler is used to generate the C or C++ bindings for all the methods which have been translated into IDL specifications, including the method query_execution. After the bindings have been generated, the corresponding C or C++ implementation code for the surrogate query_execution, query_execution_R1, query_execution_p (the original query_execution) and query_execution_R2 are inserted into the skeleton of query_execution to produce the enhanced bindings. At run-time, an activation of the method query_execution will cause the surrogate method to be executed because it is named as such. The processing of the surrogate method will call the before-rule method to carry out the query modification task, the

original query_execution method to process the subquery, and the immediate-after-rule method to do the data conversion. Thus, the mediation and query processing tasks are carried out in a distributed and active manner. This compilation approach offers the needed run-time efficiency.

4 Conclusion

In this paper, we have presented an approach to achieve distributed information mediation and query processing in a CORBA environment. The key features of the proposed approach are summarized below. First, a semantically-rich object-oriented common modeling language is used to capture the semantics of the data resources and object services of component systems. Some of these semantic properties will be lost if IDL is used to define these resources and services. Second, a high-level mediation specification language is used to explicitly specify the similarities and discrepancies of the data resources and the conversion methods needed for data conversions. Third, the information and mediation specifications using the above two languages are used to generate distributed code which is linked to the code of the component systems. Thus, much of the mediation and query processing tasks can be carried out locally and in a distributed and parallel fashion, without the overloading problem commonly seen in a centralized mediation and query processing system. Lastly, since a mediated global schema formed by a set of mediated component schemas is used instead of a traditional integrated global schema, a query issuer can use the naming, structural and semantic representations of the component schema familiar to him/her to state query and receive data from multiple data sources in his/her own view. This is more advantageous than the traditional integrated schema approach in which all users are forced to see data elements in the same way.

At the time of writing this paper, the mediation language translator, the NCL compiler and its code generation facility, and the supporting KBMS have been implemented. The generated components for distributed mediation and query processing have been tested using IBM's implementation of ORB (the SOM software package) as the communication infrastructure. Further testing of these components using different application data is under way.

References

[AMB94] J. L. Ambite, Y. Arens, C. Chee, C. N. Hsu, and C. A. Knoblock, "SIMS Manual," Working Draft, July 1994.

[BRE90] Y. Breitbart, "Multidatabase Interoperability," *SIGMOD Record*, Vol. 19, No. 3, September 1990, pp. 53-60.

[CHA91] A. Chatterjee and A. Segev, "Data Manipulation in Heterogeneous Databases," *SIGMOD Record*, Vol. 20, No. 4, December 1991, pp. 64-68.

[CHAW94] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom, "The TSIMMIS Project: Integration of Heterogeneous Information Sources," *Proceedings of IPSJ Conference*, Tokyo, Japan, October 1994, pp. 7-18.

[CHAL94] H. Chalupsky and S. C. Shapiro, "Ontological Mediation: An Analysis," Draft, Department of Computer Science, State University of New York at Buffalo, Buffalo, NY, July 1994.

[CHE88] L. Chen and P. Arbee, "Schema Integration to Support Multiple Database Access," *Bellcore, TM-STS-012948*, December 1988.

[COL91] C. Collet, M. N. Huhns and W.-M. Shen, "Resource Integration Using a Large Knowledge Base in Carnot," *IEEE Computer*, Vol. 24, No. 12, December 1991, pp. 55-62.

[FLO96] D. Florescu, L. Raschid and P. Valduriez, "A Methodology for Query Reformulation in CIS using Semantic Knowledge," *International Journal of Intelligent and Cooperative Information Systems*, special issue on Formal Methods in Cooperative Information Systems, 1996.

[GOH94] C. H. Goh, S. Madnick, and M. Siegel, "Context Interchange: Overcoming the Challenges of Large-Scale Interoperable Database Systems in a Dynamic Environment," *Proceedings of the Third Int'l Conf on Information and Knowledge Management*, November, 1994, pp. 337-346.

[HAM93] J. Hammer and D. Mcleod "An Approach to Resolving Semantic Heterogeneity in a Federation of Autonomous, Heterogeneous Database Systems," *International Journal of Intelligent and Cooperative Information Systems*, Vol.2, No. 1, 1993, pp.51-83.

[ISO92] Subcommittee 4 of ISO Technical Committee 184, "Product Data Representation and Exchange - Part 11: The EXPRESS Language Reference Manual," *ISO Document*, ISO DIS 10303-11, August 1992.

[KIM91] W. Kim and J. Seo, "Classifying Schematic and Data Heterogeneity in Multidatabase Systems," *IEEE Computer*, Vol. 24, No. 12, December 1991, pp 12-18.

[NII95] NIIIP Consortium, "NIIIP Reference Architecture: Concepts and Guidelines," NIIIP Publication NTR95-01, Jan. 1, 1995.

[SAU96] Gunter Sauter and Wolfgang Kafer, "BRIITY - A Mapping Language Bridging Heterogeneity," *Proc. of the ITG/GI/GMA Conf., Software Technology in Automation and Communication* (STAK), Munchen, Germany, March 1996.

[SHY96] Y. M. Shyy, J. Arroyo-Figueroa, S. Y. W. Su, and H. Lam, "The Design and Implementation of K: A High-level Knowledge Base Programming Language of OSAM*.KBMS", *VLDB Journal*, Vol. 5, No. 3, 1996.

[SU89] S. Y. W. Su, V. Krishnamurthy and H. Lam, "An Object Oriented Semantic Association Model (OSAM*)," Chapter 17 in *Artificial Intelligence: Manufacturing Theory and Practice*, Institute of Industrial Engineers, Industrial Engineering and Management Press, Norcross, GA, 1989, pp. 463-494.

[SU91] S. Y. W. Su and J. H. Park, "An Integrated System for Knowledge Sharing among Heterogeneous Knowledge Derivation Systems," *International Journal of Applied Intelligence*, 1, 1991, pp. 223-245.

[SU95] S. Y. W. Su, H. Lam, T. F. Yu, et al, "An Extensible Knowledge Base Management System for Supporting Rule-based Interoperability among Heterogeneous Systems," *Proceedings of the Conference on Information and Knowledge Management (CIKM)*, Baltimore, MD, November 28 - December 2, 1995, pp. 1-10.

[SU96] S. Y. W. Su, H. Lam, T. F. Yu, et al, "NCL: A Common Language for Achieving Rule-Based Interoperability among Heterogeneous Systems," *Journal of Intelligent Information Systems (JIIS)*, Special Issue on Intelligent Integration of Information, 1996, pp. 171-198.

[VEN91] V. Ventrone and S. Heiler, "Semantic Heterogeneity as a Result of Domain Evolution," *SIGMOD Record*, Vol. 20, No. 4, December 1991, pp 16-20.

[WIE92] G. Wiederhold, "Mediators in the Architecture of Future Information Systems," *IEEE Computer*, Vol. 25, No. 3, March 1992, pp. 38-49.

[WIE94] G. Wiederhold, "Interoperation, Mediation, and Ontologies," *Proceedings of the International Symposium on the Fifth Generation Computer Systems (FGCS94)*, Workshop on Heterogeneous Cooperative Knowledge-Bases, Vol. W3, ICOT, Tokyo, Japan, December 1994, pp.33-48.