Accepted for publication for the *Journal of Software and Systems.*

Context Interchange in a Client-Server Architecture

September 1993 CISL WP# 93-07

Michael Siegel*
Edward Sciore**
Stuart Madnick***

* Sloan School of Management
Massachusetts Institute of Technology
Cambridge, MA 02138

** Carroll School of Management
Boston College
Chestnut Hill, MA 02167

*** Sloan School of Management
Massachusetts Institute of Technology
Cambridge, MA 02138

Context Interchange in a Client-Server Architecture

Michael Siegel
Sloan School of Management
Massachusetts Institute of Technology
Cambridge, MA 02139
msiegel@mit.edu

Edward Sciore
Carroll School of Management
Boston College
Chestnut Hill, MA 02167
sciore@bcuxs2.bc.edu

Stuart Madnick
Sloan School of Management
Massachusetts Institute of Technology
Cambridge, MA 02139
smadnick@mit.edu

Abstract

Many organizations today are require information from diverse information systems, and government efforts are moving towards a National Information Infrastructure for sharing information. However, the systems used by these organizations lack the capabilities to meaningfully translate information between semantically heterogeneous environments. Because the meaning of data acquired from a source environment is usually different from that needed or expected in the receiver's environment, the effectiveness of information exchange is dependent upon the system's capability for *context interchange* — i.e., the representation, the exchange, the comparison, and the transformation of context knowledge.

We show in this paper a practical means for developing context interchange capabilities in a client-server architecture, using *semantic values* (i.e., data and their context) as the unit of data exchange. The key components in this architecture are the *context mediator server*, whose job is to identify and construct the semantic values being sent, determine when the exchange is meaningful, and convert the semantic values to the form required by the client (i.e., receiver); and the *data environment server*, whose job is to assist in the location, access, development and maintenance of the context knowledge for all clients and servers.

1 Introduction

Users and application developers today are expected to understand, using manuals and experience, the context of the systems that serve their portion of the enterprise. But as systems get larger, with increasing amount, types, and scope of data, such an understanding becomes overwhelming if not impossible. Our research has the goal of aiding users and developers by representing, moving, and processing the context along with the information it describes. This requires representations, models, manipulation languages, and reconciliation algorithms for context knowledge.

Conventional information system components (e.g., applications and database management systems) are designed for the exchange of simple values. However, this form of information exchange

is not sufficient, as it fails to map to real world systems where the meaning of exchanged values can change. Semantic values (data and context) [Sciore 1993], not simple values (data alone), more closely fulfill the requirements for information exchange. Because current applications and data sources are not equipped to send or receive data as semantic values, a new system architecture is needed to facilitate the exchange of semantic values among its component information systems. In this paper we describe such an architecture based on a generalized client-server system environment. We believe this approach provides a practical solution to semantic interoperability for existing and developing heterogeneous client-server information systems.

The paper is organized as follows. In Section 2 we introduce context and semantic values, and discuss their importance in effective information exchange. In Section 3 we review the component architecture in the simple source-receiver model [Siegel 1991a] and present some of the basic theory for representation and interchange of semantic values. In Section 4 we extend these results to define context interchange in a client-server architecture and discuss the requirements for two functionality servers: a context mediator server and a data environment server. Then in Section 5 we describe an implementation of the context mediator server. We consider how query processing is affected by the presence of data environment and context mediation services. Finally, we conclude by summarizing our results and examining open issues.

2 Context

Several researchers have provided different representations for data context through the use of schema-level metadata [Hsu 1991, Law 1988, McCarthy 1984, Siegel 1991a, Siegel 1991b, Yu 1990]. However, context is not simply a schema-based concept; instead, a hierarchy of contexts may exist. There may be a context for an enterprise, a database, a relation, an attribute, a data value, or any other aggregation of data objects. For example, the need for data value context to tag data with source information is demonstrated in [Wang 1990]. Context may appear at different levels for different sources — for example, the *currency* of a monetary value may be an attribute in one relation but be a schema-based definition in another. Because of these complexities in the modeling of context, there is a critical need for a well-defined metaschema model for context representation.

The most basic requirement for the use of context for multiple heterogeneous systems is the existence of common metadata vocabularies.[1] For example, shared or global ontologies [Collet 1991, Neches 1991] are being developed to define common vocabularies. An alternative approach might include negotiation among human experts [Trice 1990], leading to agreement on common terms. The existence of a common vocabulary describing the meaning of data need not be intrusive on the underlying systems. Any system in the enterprise can use the common vocabulary to develop rules (i.e., context knowledge) describing data semantics. Terminology outside of this common language must translate to the common vocabulary; otherwise comparison of data semantics will not be possible. We note that this approach is more practical than trying to agree on broad-based standards for databases. For example, it is much easier to agree on the meaning of the term "currency" and to enumerate the different acceptable currencies than to choose a standard currency in which all monetary values shall be stored.

[1] Without a common vocabulary, there would be the need for meta-metadata, and so on.

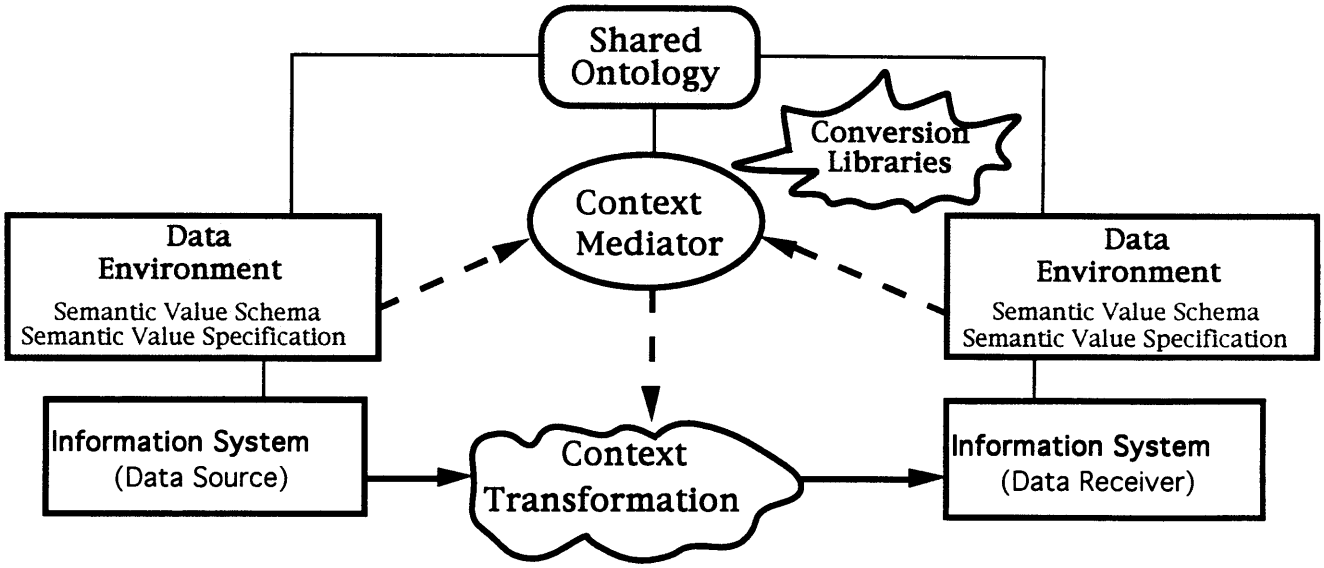

Figure 1: An Architecture for the Semantic Interoperability using Semantic Values

Our approach to representing context is to use semantic values [Sciore 1993], which are combinations of simple values and their associated properties (i.e., meta-attribute values). We represent semantic values by placing the context of a simple value next to it. For example, the following semantic value might appear in a stock market application:

1.25(Periodicity='quarterly'(FirstIssueDate='Jan. 15'), Currency='USdollars')

Here, the value *1.25* has two properties: *Periodicity* and *Currency*. The value of the former property is also a semantic value having the property *FirstIssueDate*. The semantics of *1.25* can thus be interpreted as a quarterly dividend of 1.25 US dollars with a beginning cycle of January 15th. As described in the following sections, semantic values become the unit of comparison for context interchange.

3 The Source-Receiver Architecture

The introduction of context interchange into a source-receiver system was proposed in [Sciore 1993, Siegel 1991a, Siegel 1991b]. As shown in Figure 1, the architecture contains five kinds of components: information systems, data environments, context mediators, shared ontologies, and conversion libraries.

The central component of this architecture is the *context mediator*. The context mediator is the agent that directs the exchange of values from one component information system to another, and provides services such as inter-system attribute mapping, property evaluation, and conversion. All data exchange goes through the context mediator. A context mediator is designed to function between specific component system interfaces (e.g., SQL or remote procedure calls), and is modified only when the interface is modified. The context mediator is an example of the mediator concept of [Wiederhold 1992]. This approach has two advantages: First, it is non-intrusive, in the sense

```
create table TRADES
    (CompanyName char(50),
     InstrumentType char(10),
     Exchange char(20),
     TradePrice float4
         (PriceStatus char(20),
          Currency char(15)),
     Volume int
         (Scalefactor int,
          VolumeStatus char(15)))

create table FINANCES
    (CompanyName char(50),
     Location char(40)
         (LocationGranularity char(15)),
     Revenues float4
         (Scalefactor int,
          Currency char(15)),
     Expenses float4
         (Scalefactor int,
          Currency char(15)),
     Dividend float4
         (Periodicity char(10)
              (FirstIssueDate date),
          Currency char(15)))
```

Figure 2: A Semantic Value Schema

that there is no constraint on the data models used by the component information systems. Second, it limits the number of interfaces required of each system component. The workings of the context mediator are described in more detail in Sections 4.2 and 5.

Each information system component may have an associated *data environment.* The context mediator uses the data environments of the source and receiver to determine whether a requested data exchange is possible and if so, what conversions are necessary. A data environment contains two parts: Its *semantic value schema* specifies attributes and their properties, and its *semantic value specification* specifies values for some or all of these properties. In order to create a data environment for an information system, a DBA must understand its context, the context representation language, and the common metadata vocabularies (e.g., shared ontologies).

The semantic value schema is the place where attributes and their meta-attributes are declared. Figure 2 gives a semantic value schema for a financial database containing the two relations *TRADES* and *FINANCES.* The syntax of the semantic value schema declaration is an extension of the SQL syntax for declaring relation schemas; the difference is that the semantic value schema spec-

ifies the association between each attribute and its meta-attributes. This association is achieved by placing the declaration of a meta-attribute after its associated attribute within nested parentheses. In Figure 2, the *TRADES* relation has the five base attributes *CompanyName, InstrumentType, Exchange, TradePrice,* and *Volume*; each tuple in this relation records the trading of a financial instrument (such as a company's stock) on an exchange. The *FINANCES* relation has the five base attributes *CompanyName, Location, Revenues, Expenses,* and *Dividend*; each tuple in this relation records some of the financial information about a company.

The meta-attributes defined for a relation are real attributes, in the sense that their values can be accessed by queries. However, this does not mean that meta-attributes must be explicitly stored. Meta-attribute values in a file, relation or database often have a regular, well-defined pattern. This regularity might be a consequence of behavior in the real world (e.g. "All US companies report earnings in US dollars"), business rules (e.g. "Dividends are always issued quarterly"), or characteristics of the database chosen by its DBA (e.g. "All *Volume* values are stored with a scale factor of 1000"). The rules used to compute these meta-attributes are specified in the semantic value specification of the database.

A semantic value specification consists of one or more parts, with each part applying to some set of relations (or more generally any arbitrary grouping of attributes). We call each part a *scene*. Figure 3 presents syntax for a semantic value specification containing two parts: a scene for *TRADES* and one for {*TRADES, FINANCES*}. The former scene is defined by rules, whereas the latter scene is defined by a predicate. Note that meta-attributes need not be prefixed by attribute names. The meaning of the term *"Currency = 'USdollars'"* in the latter scene of Figure 3 is that the value for the meta-attribute *Currency*, in every appropriate attribute of all relations in the group, will be *USdollars*. Scenes provide the mechanism for defining hierarchies of contexts over groupings (e.g., relational) of attributes.

The *shared ontology* component specifies *terminology mappings*. These mappings describe naming equivalences among the component information system, so that references to attributes (e.g. *Exchange* or *CompanyName*), properties (e.g. *Currency*), and their values (e.g.'USdollar') in one information system can be translated to the equivalent names in another. The development and implementation of an ontology is a complex problem, and is part of a basic goal in enterprise modeling at both a technical [Collet 1991, Neches 1991] and organizational level.

The final kind of component is the *conversion library*. A conversion library contains conversion functions; there is a global conversion library, as well as a local conversion library for each information system component. A property may have conversion functions in several libraries, and a library may have multiple conversion functions for a single property. For example, consider the property *Currency*. The global conversion library may contain a large number of conversion functions for this property, according to different dates and locations.

4 Context Interchange in Client-Server Systems

In this section we extend the source-receiver approach to a client-server architecture as shown in Figure 4. Our generalized client-server approach is typical of systems being considered or installed by leading edge information technology users. The client level contains much of the presentation software (e.g., Powerbuilder, Visual Basic), the server level contains the databases and information

create scene for TRADES **by rules**
if InstrumentType = 'equity' **and** Exchange = 'madrid'
then TradePrice.PriceStatus = 'latestClosingPrice' **and**
TradePrice.Currency = 'Pesetas';
if InstrumentType = 'equity' **and** Exchange = 'ny se'
then TradePrice.PriceStatus = 'latestTradePrice' **and**
TradePrice.Currency = 'USdollars';
if InstrumentType = 'future'
then TradePrice.PriceStatus = 'latestClosingPrice' **and**
Currency = 'USdollars';
if InstrumentType = 'equity'
then Volume.VolumeStatus = 'latestVolume' **and**
Volume.Scalefactor = 1;
if InstrumentType = 'future'
then Volume.VolumeStatus = 'closingVolume' **and**
Volume.Scalefactor = 1000;

create scene for TRADES, FINANCES **by predicate**
Currency = 'USdollars' **and** ScaleFactor = 1

Figure 3: A Semantic Value Specification with Two Scenes

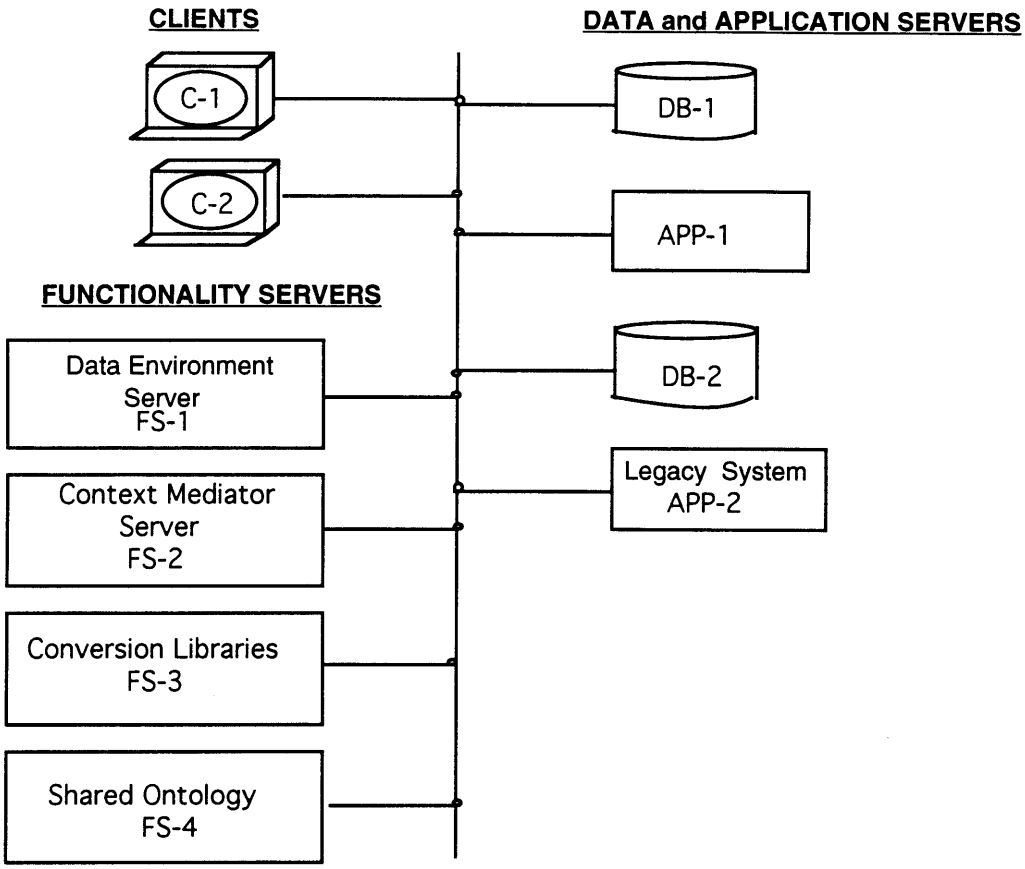

Figure 4: Client-Server Architecture with Context Mediation

application servers and a middle layer contains multiple functionality servers (e.g., the security server and name server). Many of the functionality services, excluding context interchange, are being provided by distributed environment products such as OSF's Distributed Computing Environment (DCE). Like other servers, the context mediator and data environment server may be replicated and/or distributed. For simplicity, the only functionality servers shown are those pertinent to context interchange, specifically, the context mediator, the data environment, shared ontology, and conversion library servers.

4.1 The Data Environment Server

The data environment server provides location, access, and interpretation facilities for users of the data environments. The server maintains a table with an entry for each client or server in the system, the location of its data environment, and a reference to the procedure required to access that environment. An example of a portion of such a table is shown in Figure 5. In this example table, the data environment for each server is accessed using a SQL stored procedure (e.g., SQLproc1, SQLproc2), and the environments are stored on existing database servers DB-1 and DB-2.

Component Name	Environment Location	Access Method
C-1	DB-2	SQLproc1
DB-1	DB-1	SQLproc2
APP-2	DB-1	SQLproc2

Figure 5: Environment Access Table

Our architecture allows for substantially more generality than this example shows, in several ways. First, data environments need not be relational (as presented in Section 3), and the procedure accessing a data environment may be written in other data manipulation languages. Second, the data environments for legacy systems (e.g., application and databases) and clients may be stored at other servers or even on other clients that have auxiliary storage capabilities. The data environment and context mediator servers also have associated data environments, defining the meaning of information returned by the servers, and these environments may also be stored anywhere on the system. In addition, newer system components (e.g., based on Context-SQL [Sciore 1993]) may include data environment specifications as an integral part of the client or server. The presence of a single interface to all data environments (such as the context definition language of [Siegel 1991a]) will simplify their development, manipulation, and maintenance.

The data environment server will also track changes to the environments. As described in the next subsection, this information will be used by the context mediator process to provide for consistent data exchange in a semantically evolving system.

The primary users of the data environment server are data environment builders and the context mediator server. The need to assist data environment developers suggests that the server should provide tools for context knowledge acquisition and for the design, development and maintenance of semantic value schemas and semantic value specifications. These tools should include utilities for access to ontologies, conversion libraries, and any other servers that would facilitate the development and maintenance of the data environment.

4.2 The Context Mediator Server

The *context mediator server* controls the activation and coordination of multiple context mediation processes (i.e., context mediators). The context mediator server provides these services based on system interfaces. For example, there is a context mediator for a SQL client requesting data from an SQL server and a different mediator for a menu-based client and file-based server. If a mediation process does not exist for an interface, the mediator server will attempt to link multiple processes to provide the correct interface. If no mediator can be constructed then the server returns an error. This is equivalent to saying that it is not possible to guarantee meaningful data exchange between those system components. The context mediator server maintains a reference table defining the interface(s) for each client and server. For example, client C-1 may use an SQL interface and an RPC interface, while server DB-1 uses only an SQL interface.[2]

[2]Actually, interfaces need to be more completely defined, because there are multiple types of SQL or RPC interfaces.

The context mediator server provides the following functionality:

- It recognizes the interface for each system component.
- It identifies and activates the appropriate context mediator process for a given interface.
- It coordinates the evolution of data environments and context mediation processes.

Any request made for information from a data server will invoke a call to the context mediator server. The server will create a context mediator to control the flow of information from the server to the client. After the context mediator for a client-server pair is initialized, it accesses the data environment server to locate and retrieve the appropriate data environments. These environments will be compared and, as described in Section 5, a conflict table summarizing the semantic differences between the client and the server will be constructed. The context mediator then waits for requests from the client. When a request occurs, it develops a plan for accessing the appropriate data from the data server and converting it to the environment of the client. The creation of this access plan will involve interactions with both client and data server, and may include queries to other servers such as ontologies, conversion libraries or other data servers. The optimization of this process will be complex, as the system components may have different interfaces and internal optimization capabilities. The mediator will then control the execution of the access plan, a dynamic operation whose intermediate results may cause changes to the initial plan, possibly even resulting in the addition or removal of other data servers or conversion libraries. The mediator must then deliver information in the appropriate form to the client. Depending on the client interface, if unresolved semantic conflicts occur then a failure report may be delivered to the client, which may include partial results [Siegel 1991a]. Finally, this all must occur in a system where data semantics are evolving. The context mediator server will notify the appropriate context mediator(s) when changes occur in a data environment used by that mediator. The process of evolution must produce consistent (e.g., serializable) results based on the timing of the changes to the data environment.

In the next section we will describe an example of the the functionality provided by both the context mediator server and a particular context mediator.

5 An Implementation of a Context Mediator Server

In this section we examine one particular implementation of a context mediator server and the mediation process for a particular interface. This interface consists of a relational application as the client and a relational database as the data server, using rule-based data environments and standard SQL queries. In such a system, the representation and manipulation of semantic values remains transparent to both the client and the data server.

There are several assumptions made in this implementation. First, we assume that the relations provided by the data server and viewed by the client have exactly the same base attributes — the only difference between the relations is that their environments might specify different meta-attributes and meta-attribute values. Second, we assume that all meta-attributes are derived and that data environments are rule-based, with the antecedent of each rule composed of restrictions on base attributes and the consequent assignment of values to meta-attributes; a detailed discussion of

```
create scene for TRADES by rules
    if Exchange = 'madrid'
        then TradePrice.PriceStatus = 'latestClosingPrice' and
             TradePrice.Currency = 'US dollars';
    if Exchange = 'nyse'
        then TradePrice.PriceStatus = 'latestTradePrice' and
             TradePrice.Currency = 'USdollars';
    if TRUE
        then Volume.VolumeStatus = 'closingVolume' and
             Volume.Scalefactor = 1;
```

Figure 6: A Client's Data Environment

the structure of such rules appears in [Madero 1992, Siegel 1991a]. Third, we assume that the data server and client agree on the meaning of attribute names, properties, and values, and so there is no need for a shared ontology server.

As a running example, we assume that the semantic value schemas for both client C-1 and the data server DB-2 in Figure 4 are the semantic values in the *TRADES* relation of Figure 2; the data server uses the data environment of Figure 3 and the client uses the data environment of Figure 6.

When the context mediator server detects the need to connect the client with the data server, it starts an appropriate context mediator process. This mediator then takes responsibility for all communication between the client and data server.

The first task of the newly spawned context mediator is to preprocess the data environments of the data server and the client, creating a *conflict table.* Each row of the conflict table describes when and how a value from the data server should be converted to the environment of the client. Details of this construction appear in [Madero 1992, Sciore 1993, Siegel 1991a]. Figure 7 depicts the conflict table for our running example. The first rule of the table states that if a source tuple is an equity traded on the Madrid exchange, then the value of its *TradePrice* attribute must be converted from Pesetas to US dollars; this rule resulted from comparing the first rules in the two data environments. Similarly, the second row of the table results from a conflict between rule 3 of the data server's environment and rule 2 of the client's environment; the third row of the conflict table results from rule 4 of the data server's environment and rule 3 of the client's environment; and the fourth row of the conflict table results from rule 5 of the data server's environment and rule 3 of the client's environment.

Intuitively, each row of the conflict table describes how a tuple from the server database should be converted to the environment of the client. In particular, let t be a database tuple. If t satisfies none of the constraints of the rows in the conflict table, then t can be passed to the client without any conversions. If t satisfies one or more of the constraints, then appropriate conversions must occur. For example, suppose that t has the values $t.InstrumentType$ = 'equity' and $t.Exchange$ = 'madrid'. Then t satisfies the constraints of rows 1 and 3 of the conflict table, and so $t.TradePrice$ must be converted from Pesetas to US dollars and $t.Volume$ must be converted from the latest volume to the closing volume.

Constraint	Attribute	Meta-Attr	DBValue	AppValue
InstrumentType='equity' **and** Exchange='madrid'	TradePrice	Currency	'Pesetas'	'USdollars'
InstrumentType='future' **and** Exchange='nyse'	TradePrice	PriceStatus	'latestClosingPrice'	'latestTradePrice'
InstrumentType='equity'	Volume	VolumeStatus	'latestVolume'	'closingVolume'
InstrumentType='future'	Volume	ScaleFactor	1000	1

Figure 7: A Conflict Table

After the pre-processing stage is finished, the context mediator is ready to process queries from the client. This activity is an extension of what would be performed by a standard query processor, and includes query transformation and optimization. Suppose that the application poses a query Q. After intercepting the query, the context mediator performs the following tasks:

1. It transforms the query into relational algebra in the standard way, creating a query tree for it;
2. it replaces each reference to relation R by the expression $cvt_{E'_R}(R)$, where E'_R is the client's data environment for R and *cvt* is a new relational operator (see below) that performs conversion;
3. it performs transformations on the query tree for the purposes of optimization;
4. it evaluates the query tree.

Details of these steps are discussed in the following paragraphs.

The first step is straightforward. The second step introduces conversion into the query. The client's query is based on the assumption that the data is stored according to its environment. The simplest way to process this query, therefore, is to first convert all relations mentioned in the query to the client's environment. That is the function of the *cvt* operator. Formally, if T is a relation (or an expression producing a relation), then the expression $cvt_E(T)$ produces a relation having the same tuples as T, except that they are converted from the environment of T to the environment E.

This strategy of converting entire relations is straightforward to implement, but is not especially efficient. For example, it might be more appropriate for selections to occur first, so that there are fewer tuples to convert. The determination of when conversions should occur is a form of the general optimization problem. In particular, conversions can be delayed by transforming the query tree so that *cvt* nodes are higher up. The context mediator performs this transformation by utilizing certain algebraic equivalences involving *cvt* and the other relational operators. The simplest equivalence involves union. The *cvt* operator can move in and out of unions unconditionally as follows:

$$cvt_E(R) \cup cvt_E(S) \Longleftrightarrow cvt_E(R \cup S)$$

Similar equivalences are also true for set difference and intersection. The product operator is also straightforward. Assuming that the schemas of R and S are disjoint, the following transformation holds unconditionally:

$$cvt_{E1}(R) \times cvt_{E2}(S) \Longleftrightarrow cvt_{E1 \cup E2}(R \times S)$$

Equivalences involving the selection operator are the most important and also the most difficult. We need the following terminology: We say that an environment is *simple* for attribute A if for all meta-attributes $A.M$, the value of $A.M$ is the same in every possible tuple of the relation. Given an attribute A, it is possible to find predicates $\{P_1, \ldots, P_k\}$ that partition R and for which both the data server and client environments are simple for A on each partition. These predicates are determined by comparing the rule sets of the two environments, using an algorithm similar to the one that creates the conflict table. Once the partition is found, the following equivalence holds:

$$\sigma_{A=c}(cvt_E(R)) \Longleftrightarrow \bigcup cvt_E(\sigma_{P_i \wedge (A=c_i)}(R))$$

where c_i is the value that results from converting c from E to the simple environment of $\sigma_{P_i}(R)$. An example transformation using this equivalence appears below.

The final step is for the context mediator to evaluate the query tree. The context mediator is responsible for doing conversions, and so the query tree must be evaluated in stages. In particular, each subtree of the query tree rooted by a *cvt* node defines a temporary relation. The query corresponding to each subtree is sent to the data server, and the resulting tuples are converted and stored in the temporary relation; a reference to this temporary relation then replaces the subtree in the query tree. The result of the final query is sent to the client. The only case when no temporary relations are necessary is when the only conversion appears at the top of the query tree. In this case, the context mediator can send the rest of the query to the data server, convert the resulting tuples as they are returned, and then forward the converted tuples to the client.

For an example of this process, suppose that the client poses the following query:

select *
from TRADES
where InstrumentType='equity' **and** TradePrice>100

Because the client uses the data environment of Figure 6, this query requests those equities having a latest closing price of more than 100 US dollars. The context mediator proceeds as follows. Let E_S be the environment of the data server and E_C be the environment of the client. The query tree after step 2 is shown in Figure 8. In step 3, the algebraic equivalences are used to push the *cvt* operator outside the selections. The following two predicates partition the relation:

P_1 = (Exchange = 'madrid')
P_2 = (Exchange $\neq$ 'madrid')

The resulting selection for P_1 is *TradePrice* > *12000*, assuming 120 Pesetas to the US dollar. The resulting selection for P_2 is *TradePrice* > *100*, because no currency conversion is required. The *cvt* node can be pushed past the selection on *InstrumentType* trivially, because no conversions are necessary. The query tree after these transformations is shown in Figure 9.

Standard query optimizations can then be used to push the upper selection down into the union and merge the sequential selections, producing the final query tree of Figure 10. The subquery below the *cvt* node is then sent to the data server, and the resulting tuples are converted and sent to the client.

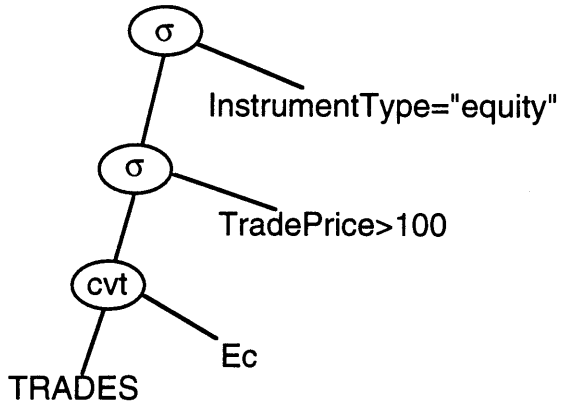

Figure 8: The query tree before optimization

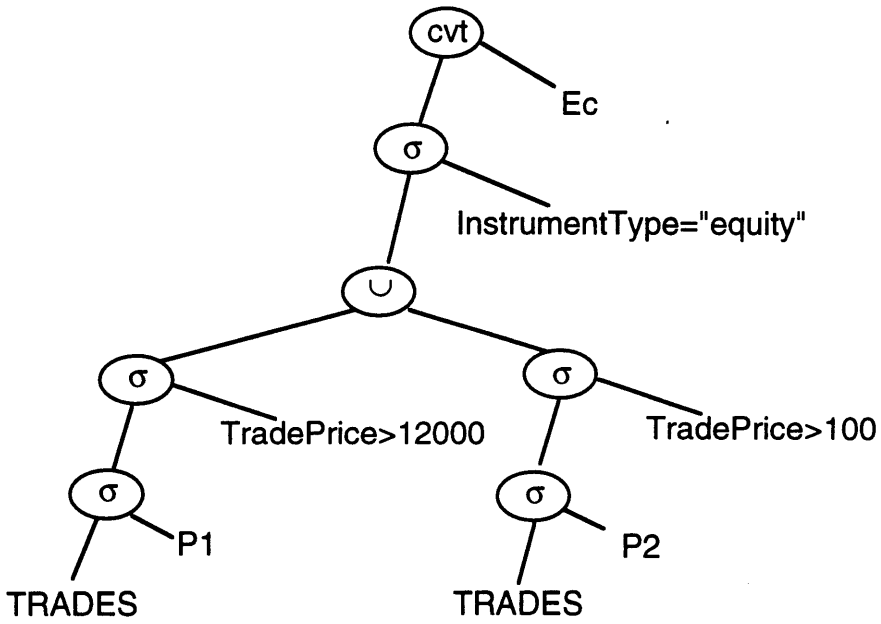

Figure 9: The query tree after the first transformation

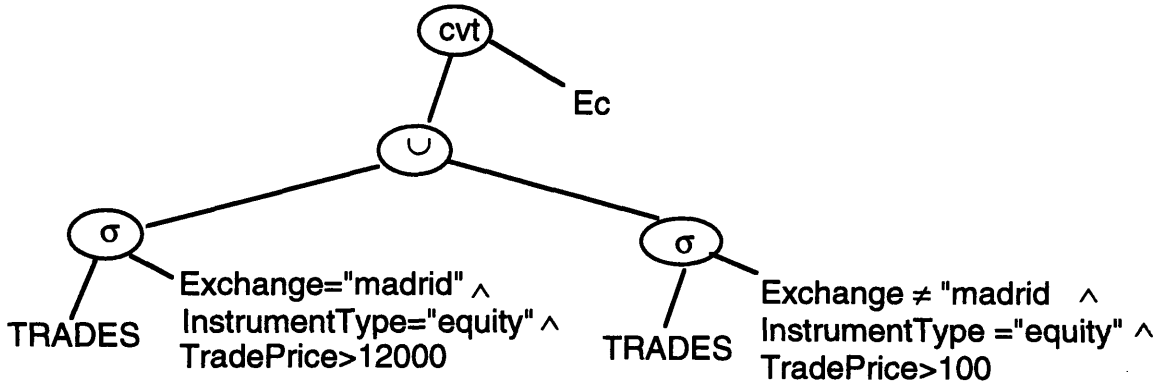

Figure 10: The final query tree

One aspect of the context mediator that we have not discussed is the way in which the *cvt* operator is implemented. In general, the context mediator uses the conflict table to generate conversion plans efficiently and determine when conversions are impossible without having to go to the database. Details of this process can be found in [Sciore 1993].

6 Conclusions

In this paper we present a practical approach to including context interchange capabilities in a client-server architecture. Our approach is based on the use of a data environment to record context knowledge for both clients and data servers. We propose a structure for the data environments based on the use of semantic values as the unit of data comparison. This approach is non-intrusive to existing databases and applications, as the data environments remain external to existing systems. We introduce two server components into our architecture: a context mediator server and a data environment server. Both of these components may be replicated and/or distributed. The data environment server provides location, access, development and maintenance utilities for working with each systems environment. The context mediator server provide context mediation capabilities for the exchange of information between each pair of systems. We describe the role of the context mediator in query processing. We show how conversion can be viewed as a relational operation and present query optimization possibilities based on the evaluation of an extended-relational-algebraic query tree. Our current implementation effort is providing for context interchange capabilities as an add-on to existing commercial technologies that provide client-server computing.

Our work suggests several areas for future research. First, we need to examine performance issues related to context mediation and the proposed architecture. We believe there are several approaches that may be used for optimization of the context interchange process. This will become an important area of research as context mediators are developed for clients and servers with a variety of internal capabilities. Second, we need more practical experience in acquiring context knowledge and developing data environments, in order to develop tools and methodologies for context knowledge acquisition and maintenance. Third, we need to consider the possibility of using multiple, cooperating mediators. In addition, considerable research is needed to develop strategies

for building mediators. Fourth, it is necessary to define algorithms that the mediator can use to plan conversions; this planning can be nontrivial when the behavior of a conversion is more complex than simply changing a meta-attribute from one value to another. We need to examine the role of conversion functions, the use of complex conversions, and the location and selection of conversion libraries and functions (e.g., local vs. global). Fifth, consistency conditions must be defined for evolution of data semantics and context mediator operation. Finally, the approach in this paper needs to be extended to a more general distributed computing environment.

Acknowledgments

This work is supported in part by NSF grant IRI-90-2189, ARPA grant F30602-93-C-0160, and the International Financial Services Research Center at MIT

References

[Collet 1991] C. Collet, M. Huhns, and W. Shen. Resource integration using an existing large knowledge base. Technical Report ACT-OODS-127-91, MCC, 1991.

[Hsu 1991] C. Hsu, M. Bouziane, L. Rattner, and L. Yee. Information resources management in heterogeneous, distributed environments. *IEEE Transactions on Software Engineering*, 17(6):604–625, June 1991.

[Law 1988] M. H. Law. *Guide to Information Resource Dictionary System Applications: General Concepts and Strategic Systems Planning.* 500-152. National Bureau of Standards, 1988.

[Madero 1992] F. Madero. Rule-based mediator implementation for solving semantic conflicts in SQL. Master's thesis, MIT, September 1992.

[McCarthy 1984] J. McCarthy. Scientific information = data + meta-data. In *Database Management: Proceedings of the Workshop November 1-2, U.S. Navy Postgraduate School, Monterey, California.* Department of Statistics Technical Report, Stanford University, 1984.

[Neches 1991] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12(3):37–56, 1991.

[Sciore 1993] E. Sciore, M. Siegel, and A. Rosenthal. Using semantic values to facilitate semantic interoperability among heterogeneous information systems. *In Submission to Transactions on Database Systems*, 1993.

[Siegel 1991a] M. Siegel and S. Madnick. A metadata approach to resolving semantic conflicts. In *Proceedings of the 17th International Conference on Very Large Data Bases*, pages 133–145, Barcelon, Spain, September, 1991.

[Siegel 1991b] M. Siegel and S. Madnick. Metadata requirements in resolving semantic heterogeneities. *Sigmod Record Special Issue on Semantic Heterogeneity*, 1991.

[Trice 1990] A. Trice. *Facilitating Consensus Knowledge Aquisition.* PhD thesis, Massachusetts Institute of Technology, 1990.

[Wang 1990] R. Wang and S. Madnick. Data-source tagging. In *Proceeding of the Very Large Database Conference*, pages 519–538, Brisbane, Australia, 1990.

[Wiederhold 1992] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, March 1992.

[Yu 1990] C. Yu, W. Sun, S. Dao, and D. Keirsey. Determining relationships among attributes for interoperability of multi-database systems. In *Position Papers: Workshop on Multidatabases and Semantic Interoperability*, November 2-4, 1990.