Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Systems

by

Cheng Hian Goh

Submitted to the Sloan School of Management
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Jan 1997

Author .....
Sloan School of Management
Dec 5, 1996

Certified by .....
Stuart E. Madnick
John Norris Maguire Professor of Information Technology
Thesis Supervisor

Accepted by .....
Gordon M. Kaufman
Chairman, Departmental Committee on Graduate Students

Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Systems

by

Cheng Hian Goh

Submitted to the Sloan School of Management
on Dec 5, 1996, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

The COntext INterchange (COIN) strategy [Sciore et al., 1994, Siegel and Madnick, 1991] presents a novel perspective for mediated data access in which semantic conflicts among heterogeneous systems are not identified a priori, but are detected and reconciled by a *Context Mediator* through comparison of *contexts* associated with any two systems engaged in data exchange. In this paper, we present a *formal characterization* and *reconstruction* of this strategy in a COIN *framework*, based on a *deductive object-oriented* data model and language called COIN. The COIN framework provides a logical formalism for representing data semantics in distinct contexts. We show that this presents a well-founded basis for reasoning about semantic disparities in heterogeneous systems. In addition, it combines the best features of *loose-* and *tight-coupling* approaches in defining an integration strategy that is *scalable*, *extensible* and *accessible*. These latter features are made possible by allowing complexity of the system to be harnessed in small chunks, by enabling sources and receivers to remain loosely-coupled to one another, and by sustaining an infrastructure for data integration. The feasibility and features of this approach have been demonstrated in a prototype implementation which provides mediated access to traditional database systems (e.g., Oracle databases) as well as semi-structured data (e.g., Web-sites)

Thesis Supervisor: Stuart E. Madnick
Title: John Norris Maguire Professor of Information Technology

Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Systems

by

Cheng Hian Goh

Submitted to the Sloan School of Management
on Dec 5, 1996, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

The COntext INterchange (COIN) strategy [Sciore et al., 1994, Siegel and Madnick, 1991] presents a novel perspective for mediated data access in which semantic conflicts among heterogeneous systems are not identified a priori, but are detected and reconciled by a *Context Mediator* through comparison of *contexts* associated with any two systems engaged in data exchange. In this paper, we present a *formal characterization* and *reconstruction* of this strategy in a COIN *framework*, based on a *deductive object-oriented* data model and language called COIN. The COIN framework provides a logical formalism for representing data semantics in distinct contexts. We show that this presents a well-founded basis for reasoning about semantic disparities in heterogeneous systems. In addition, it combines the best features of *loose-* and *tight-coupling* approaches in defining an integration strategy that is *scalable*, *extensible* and *accessible*. These latter features are made possible by allowing complexity of the system to be harnessed in small chunks, by enabling sources and receivers to remain loosely-coupled to one another, and by sustaining an infrastructure for data integration. The feasibility and features of this approach have been demonstrated in a prototype implementation which provides mediated access to traditional database systems (e.g., Oracle databases) as well as semi-structured data (e.g., Web-sites)

Thesis Supervisor: Stuart E. Madnick
Title: John Norris Maguire Professor of Information Technology

Contents

List of Figures

There is nothing more practical than a good theory.

– Kurt Lewin

Chapter 1

Introduction

Almost every statement we make is imprecise and hence is meaningful only if understood with reference to an underlying *context*[1] which embodies a number of hidden assumptions. This anomaly is amplified in databases due to the gross simplifications that we made in creating a database schema. For example, a database may record the fact

```
salary('Jones', 2000)
```

without ever explaining what "2000" means (what currency and scale-factor is used?), what is the periodicity (is this the daily, weekly, or monthly wage?), or what constitutes the person's salary (does it include year-end bonuses? what about overtime pay?). The real problem occurs when data sources and receivers maintain different assumptions about the data which are being exchanged: a receiver formulates a query and interprets the answers returned in a certain context, whereas the query is executed by source(s) which most likely provide answers in a completely different context. Under these circumstances, *physical connectivity* (the ability to exchange bits and bytes) does not necessarily lead to *logical connectivity* (the ability to exchange meaningful information). This problem is traditionally referred to as the need for *se-*

[1] **context**: *n.* **1.** The part of a written or spoken statement in which a word or passage at issue occurs and that often specifies its meaning. **2.** The circumstances in which a particular event occurs; situation. (`The American Heritage Dictionary`).

mantic interoperability among *autonomous* and *heterogeneous* systems [Scheuermann et al., 1990, Sheth and Larson, 1990, Hurson et al., 1994].

This Thesis describes a novel approach, called *Context Interchange* (COIN), for achieving semantic interoperability among heterogeneous sources *and* receivers. The COIN strategy described in this paper drew its inspiration from earlier work reported in [Siegel and Madnick, 1991, Sciore et al., 1994]. Specifically, we share the basic tenets that

- the *detection* and *reconciliation* of semantic conflicts are system services which are provided by a *Context Mediator*, and should be transparent to a user; and
- the provision of such a mediation service requires only that the user furnish a logical (declarative) specification of *how data are interpreted* in sources and receivers, and *how conflicts, when detected, should be resolved*, but *not what conflicts exists a priori* between any two systems.

These insights are novel because they depart from classical integration strategies which either require users to engage in the detection and reconciliation of conflicts (in the case of *loosely-coupled systems*; e.g., MRDSM [Litwin and Abdellatif, 1987], VIP-MDBMS [Kuhn and Ludwig, 1988]), or insist that conflicts should be identified and reconciled, a priori, by some system administrator, in one or more shared schemas (as in *tightly-coupled systems*; e.g., Multibase [Landers and Rosenberg, 1982], Mermaid [Templeton et al., 1987]).

Unfortunately, as interesting as these ideas may be, they could remain as vague musings in the absence of a formal conceptual foundation. One attempt at identifying a conceptual basis for Context Interchange is the *semantic-value model* [Sciore et al., 1994], where each data element is augmented with a property-list which defines its *context*. This model, however, continues to be fraught with ambiguity. For example, it relied on implicit agreement on what the modifiers for different attributes are, as well as what conversion functions are applicable for different kinds of conflicts, and is silent on how different conversion definitions can be associated with distinct contexts. Defining the semantics of data through annotations attached to individual

data elements tend also to be cumbersome, and there is no systematic way of promoting the sharing and reusing of the semantic representations. At the same time, the representational formalism remains somewhat detached from the underlying conflict detection algorithm (the *subsumption algorithm* [Siegel and Madnick, 1991]). Among other problems, this algorithm requires conflict detection to be done on a pairwise basis (i.e., by comparing the context definitions for two systems at a time), and is non-committal on how a query plan for multiple sources can be constructed based on the sets of pair-wise conflicts. Furthermore, the algorithm limits meta-attributes to only a single-level (i.e., property lists cannot be nested), and are not able to take advantage of known constraints for pruning off conflicts which are guaranteed never to occur.

1.1 Summary of Contributions

We have two (intertwined) objectives in this paper. First, we aim to provide a formal foundation for the Context Interchange strategy that will not only rectify the problems described earlier, but also provide for an integration of the underlying representational and reasoning formalisms. Second, the deconstruction and subsequent reconstruction[2] of the Context Interchange approach described in [Siegel and Madnick, 1991, Sciore et al., 1994] provides us with the opportunity to address the concern for integration strategies that are scalable, extensible and accessible.

Our formal characterization of the Context Interchange strategy takes the form of a COIN *framework*, based on the COIN data model, which is a customized subset of the deductive object-oriented model called Gulog[3] [Dobbie and Topor, 1995]. COIN is a "logical" data model in the sense that it uses logic as a formalism for representing knowledge and for expressing operations. The logical features of COIN provide us with a well-founded basis for making inferences about semantic disparities that exist

[2]In this *deconstruction*, we tease apart different elements of the Context Interchange strategy with the goal of understanding their contributions individually and collectively. The *reconstruction* examines how the same features (and more) can be accomplished differently within the formalism we have invented.

[3]Gulog is itself a variant of *F-logic* [Kifer et al., 1995].

among data in different contexts. In particular, a COIN framework can be translated to a *normal program* [Lloyd, 1987] (equivalently, a $Datalog^{neg}$ program) for which the semantics is well-defined, and where sound computational procedures for query answering exist. Since there is no real distinction between factual statements (i.e., data in sources) and knowledge (i.e., statements encoding data semantics) in this logical framework, both queries on data sources (*data-level queries*) as well as queries on data semantics (*knowledge-level queries*) can be processed in an identical manner. As an alternative to the classic deductive framework, we investigate the adoption of an *abductive framework* [Kakas et al., 1993] for query processing. Interestingly, although abduction and deduction are "mirror-images" of each other [Denecker and Schreye, 1992a], the *abductive answers*, computed using a simple extension to classic SLD-resolution leads to *intensional answers* as opposed to *extensional answers* that would be obtained via deduction. Intensional answers are useful in our framework for a number of conceptual and practical reasons. In particular, if the query is issued by a "naive" user under the assumption that there are no conflicts whatsoever, the intensional answer obtained can be interpreted as the corresponding *mediated query* in which database accesses are interleaved with data transformations required for mediating potential conflicts. Finally, by checking the consistency of the abducted answers against known integrity constraints, we show that the abducted answer can be greatly simplified, demonstrating a clear connection to what is traditionally known as *semantic query optimization* [Chakravarthy et al., 1990].

As much as it is a logical data model, COIN is also an "object-oriented" data model because it adopts an "object-centric" view of the world and supports many of the features (e.g., object-identity, type-hierarchy, inheritance, and overriding) commonly associated with object-orientation. The standard use of abstraction, inheritance, as well as *structural* and *behavioral inheritance* [Kifer et al., 1995] present many opportunities for sharing and reuse of semantic encodings. Conversion functions (for transforming the representation of data between contexts) can be modeled as *methods* attached to types in a natural fashion. Unlike "general purpose" object-oriented formalisms, we make some adjustments to the structure of our model by distinguishing

between different kinds of objects which have particular significance for our problem domain. In particular, we introduce the notion of *context-objects*, described in [McCarthy, 1987], as reified representations for collections of statements about particular contexts. This allows context knowledge to be defined with a common reference point and is instrumental in providing a structuring mechanism for making inferences across multiple theories which may be mutually inconsistent.

The reconstruction of the Context Interchange strategy allows us to go beyond the classical concern of "non-intrusion", and provides a formulation that is scalable, extensible and accessible [Goh et al., 1994]. By *scalability*, we require that the complexity of creating and administering (maintaining) the interoperation services should not increase exponentially with the number of participating sources and receivers. *Extensibility* refers to the ability to incorporate changes in a graceful manner; in particular, local changes should not have adverse effects on other parts of the larger system. Finally, *accessibility* refers to how the system is perceived by a user in terms of its ease-of-use and flexibility in supporting different kinds of queries.

The above concerns are addressed in two ways in the *reconstructed* Context Interchange strategy[4]. Provisions for making sources more accessible to users is accomplished by shifting the burden for conflict detection and mediation to the system; supporting multiple paradigms for data access by supporting queries formulated directly on sources as well as queries mediated by views; by making knowledge of disparate semantics accessible to users by supporting knowledge-level queries and answering with intensional answers; and by providing feedback in the form of mediated queries. Scalability and extensibility are addressed by maintaining the distinction between the representation of data semantics as is known in individual contexts, and the detection of potential conflicts that may arise when data are exchanged between two systems; by the structural arrangement that allow data semantics to be specified with reference to complex object types in the domain model as opposed to annotations tightly-coupled in the individual database schemas; by allowing multiple systems with

[4]For the sake of brevity, further references made to the Context Interchange strategy from this point on refers to this reconstruction unless otherwise specified.

distinct schemas to bind to the same contexts; by the judicious use of object-oriented features, in particular, inheritance and overriding in the type system present in the domain model; and by sustaining an infrastructure for data integration that combines these features. As a special effort in providing such an infrastructure, we introduce a *meta-logical* extension of the COIN framework which allows sets of context axioms to be "objectified" and placed in a hierarchy, such that new and more complex contexts can be derived through a hierarchical composition operator [Brogi and Turini, 1991]. This mechanism, coupled with type inheritance, constitutes a powerful approach for incorporating changes (e.g., the addition of a new system, or changes to the domain model) in a graceful manner.

Finally, we remark that the feasibility and features of this approach have been demonstrated in a prototype implementation which provides mediated access to traditional database systems (e.g., Oracle databases) as well as semi-structured data (e.g., Web-sites)[5].

1.2 Thesis Outline

The rest of this Thesis is organized as follows. **Chapter 2** provides a summary of the data integration literature with a taxonomy of *data conflicts*, followed by a brief survey of the various research efforts aimed at overcoming these problems. We contend, however, that existing solutions are not adequate and suggests that a viable strategy must go beyond "non-intrusiveness" to facilitate the construction of interoperable systems which are *scalable*, *extensible* and *accessible*.

Chapter 3 paves the way for the rest of the Thesis by exemplifying the features of the Context Interchange strategy through the use of examples. This provides an overview of *what* is being accomplished, *how* this is aided by the underlying COIN data model, as well as *why* the various features are useful. Technical discussion is deliberately kept to the minimal so that this material can be accessible to a wider

[5]This prototype is accessible from any WWW-client (e.g., Netscape Browser) and can be demonstrated upon request.

audience. In so doing, we hope the benefits of the proposed integration approach can be appreciated by even the casual reader who may not want to be encumbered by the technical material.

Chapter 4 begins with a review of deductive, object-oriented, and deductive object-oriented data models. This is followed by a discussion on formalizing the notion of context, first introduced in [McCarthy, 1987]. We then describe the *structure* and *language* of the COIN *data model*, which forms the basis for the formal characterization of the Context Interchange strategy in a COIN *framework*. For didactical reasons, we shall first introduce the COIN framework without considerations for inheritance among context theories, and address the latter separately as an extension.

Chapter 5 describes the abductive inference approach to query answering in the COIN framework. We introduce the subject of abduction with a brief survey of the literature; in particular, we describe the *abductive framework* as well as a simple computation procedure called *SLD+Abduction* [Cox and Pietrzykowski, 1986], which is being adapted for our purposes. As pointed out by various researchers, there is an interesting duality relationship between abduction and deduction: we illustrate the intuition underlying this with an example. We proceed to show that query answering in a COIN framework is in fact isomorphic to the same in an abductive framework. Interestingly, consistency requirements in abduction provides a natural means for integrating integrity constraints which amounts to performing classical *semantic query optimization* [Chakravarthy et al., 1990].

Chapter 6 presents details on the design and implementation of the Context Interchange Prototype. The Context Mediator (the key component responsible for rewriting a user query to a mediated one) is implemented as part of testbed which includes a frontend for query formulation, gateways to relational databases, and wrappers for semi-structured documents served by Web-sites. The Domain Model and Context Axioms for the motivational example (as presented in Chapter 3) has been captured along with several other data sources. This allows one to experiment with different types of queries to gain a better understanding of the features of our integration approach.

Finally, **Chapter 7** concludes the Thesis with a summary of our contributions and describes some important research questions which we have not been able to address in this Thesis due to resource limitations.

Chapter 2

Literature Survey

Throughout this Thesis, we use the term "heterogeneous information systems" to refer to data *sources* and data *receivers* which are cooperating to share information in the form of structured data. *Sources* refer to databases, data feeds, and other applications which provide data upon requests; *receivers* refer to users, consolidating databases (e.g., data warehouses), and applications that make these requests. For brevity, we will refer to both of these as *component systems* when the distinction is not important. Component systems in "heterogeneous information systems" are characterized by *autonomy* and *heterogeneity*, thus setting the latter apart from traditional information systems which are centrally administered to maintain an acceptable level of homogeneity by virtue of design[1].

In their report on the NSF workshop on Heterogeneous Database Systems in 1989, Scheuermann et al. [1990] noted that there are different types (degrees) of autonomy, each having different implications:

- *Design autonomy* refers to the capability of a database system to choose its own information, data model, and implementation procedures;

[1]For most purposes, our characterization of "heterogeneous information systems" is not unlike that of "heterogeneous databases" [Scheuermann et al., 1990], "federated database systems" [Sheth and Larson, 1990], or "multidatabase systems" [Bright et al., 1992] as are described in the literature. A notable exception is that we accord the same level of autonomy to both sources and receivers: i.e., disparate users and applications are equally likely to have different expectations of how data is presented and interpreted and, like sources, are resilient to changes (for example, in the case of legacy applications, changes in structure or semantics may entail code changes).

- *Communication autonomy* refers to the capability of a system to decide with what other systems to communicate and what information to exchange with them;

- *Execution autonomy* refers to the ability of a system to decide how and when to execute requests received from another system.

Design autonomy leads to various types of heterogeneity. It is common practice to distinguish between *data heterogeneity* and *system heterogeneity*. The first refers to the different ways in which data is organized or interpreted in disparate systems; the latter is concerned with differences in data model, data manipulation language, concurrency control mechanism, and so forth. Clearly, design autonomy (and resulting heterogeneity) constitutes a major obstacle in accomplishing *semantic interoperability*, or the meaningful exchange of information, among disparate systems. Communication and execution autonomy on the other hand pose many new challenges for query processing and optimization which are distinct from those faced in distributed query processing. These latter issues have been discussed in [Lu et al., 1992] but are otherwise outside the scope of this Thesis.

2.1 Data Heterogeneity

Roughly speaking, we can distinguish between three types of data conflicts. Of these, the first two — *schematic conflicts* [Kim and Seo, 1991, Krishnamurthy et al., 1991], *semantic conflicts* [Sheth and Kashyap, 1992, Naiman and Ouskel, 1995, Garcia-Solaco et al., 1996] — have been well-documented in the literature (though there is little consensus of what each encompasses). In most instances, the distinction between the two can be characterized by differences in "structure" ("how are the data logically organized?") versus that of "interpretation" ("what do the data mean?"). This distinction however is not always crisp, since the logical organization of data often conveys semantic information. A third category of conflicts, which we shall refer to as *intensional conflicts*, are concerned with the differences over the *contents*

of information which are present (or expected) in distinct systems. The remainder of this section introduces a taxonomy (as illustrated in Figure 2-1) which provides a synthesis of the literature on this subject.

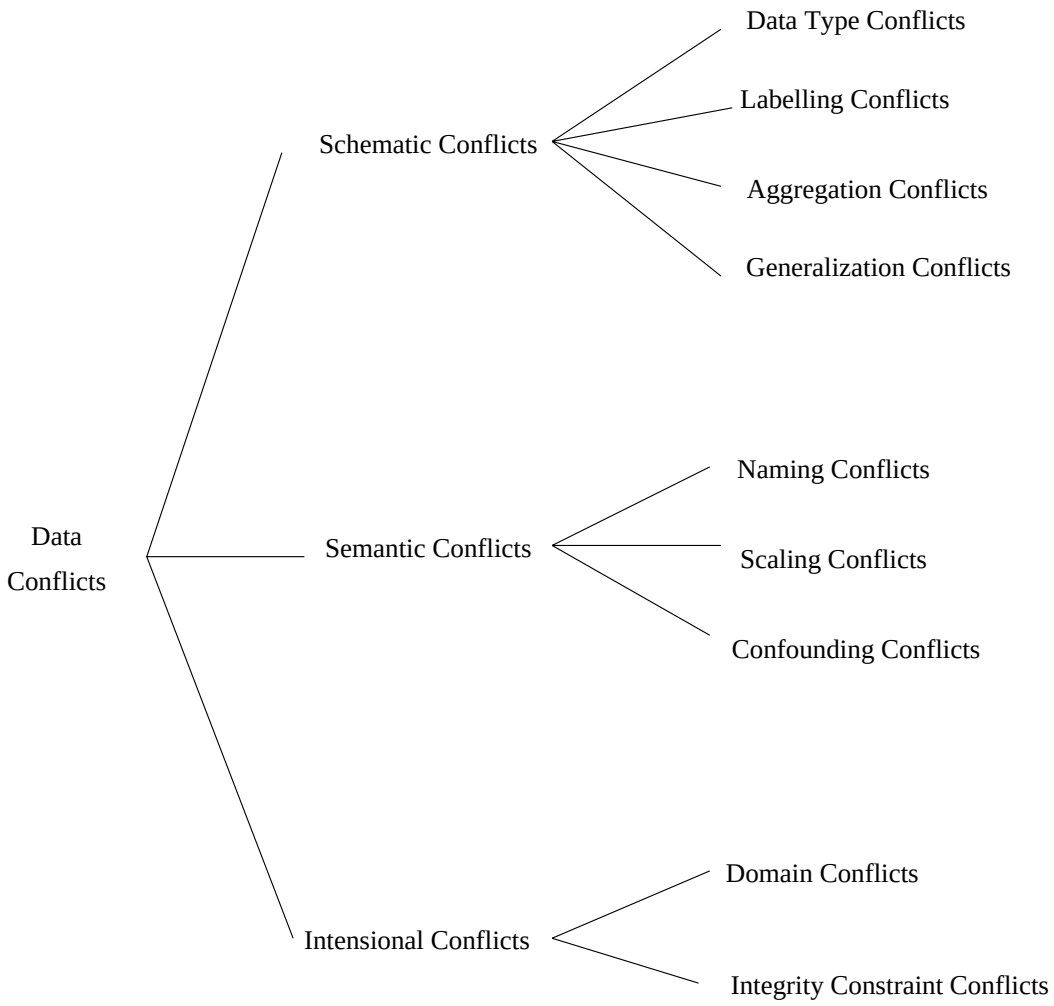

Figure 2-1: A taxonomy of data conflicts proposed in this Thesis.

Schematic Heterogeneity

Four distinct categories of conflicts are often identified with schematic heterogeneity.

Data Type Conflicts

Data Type conflicts refer to the use of different primitive system types in the representation of data values. For example, dates may be represented as strings, or, in the case of some DBMSs, instances of the primitive type "Date". These differences may stem from arbitrary choices of different system designers (i.e., design autonomy), or they may be due to the fact that a given data type (in this case, "Date") is not supported by the underlying system. In some instances, these constraints may be more subtle (e.g., as when the underlying system impose an upper limit on the precision of a stored value), in which case the conflicts, though real, may not be readily apparent.

Labelling Conflicts

Labelling Conflicts refer to synonyms and homonyms as occurring among schema elements: i.e., the same attribute may be referred to by different labels in two distinct sources, or conversely, the same label may be used in identifying distinct attributes of an entity. In the case of the relational data model, this translates to conflicts between table names and/or attribute names [Kim and Seo, 1991]. Conflicts of this kind are easily resolved through renaming, such as via a view definition. Other strategies include the use of some *renaming operator* [Motro, 1987], or the definition of *superfunctions* [Dayal and Hwang, 1984].

Aggregation Conflicts

Aggregation Conflicts stem from different design choices concerning how data should be clustered to form the attributes of an entity being modeled [Smith and Smith, 1977]. For example, a designer may choose to model data concerning stocks by clustering them around each instrument, or modeling transactions on a daily basis in which case the different instruments are attributes. These different choices lead to the juxtapositioning of data and meta-data as shown in Figure 2-2.

Database A

relation *Jan0196*

StkCode	TradePrice
HPP	30.10
BBM	40.20
⋮	⋮

relation *Jan0296*

StkCode	TradePrice
HPP	30.50
BBM	41.00
⋮	⋮

Database B

relation *HPP*

Date	TradePrice
01/01/96	30.10
01/02/96	30.50
⋮	⋮

relation *BBM*

Date	TradePrice
01/01/96	40.20
01/02/96	41.00
⋮	⋮

Figure 2-2: Schematic conflicts resulting from the different design decisions concerning how data should be aggregated.

Database A

relation *Employees*

Name	Department	Designation
Jones	production	manager
Simpson	development	engineer
⋮	⋮	⋮

Database B

relation *Managers*

Name	Department
Jones	production
⋮	⋮

relation *Engineers*

Name	Department
Simpson	development
⋮	⋮

Figure 2-3: Schematic conflicts resulting from different design choices concerning how "generalization" is applied.

Generalization Conflicts

Generalization Conflicts differ from aggregation conflicts in that the design choices involve determine how different "entity types" relate to one another through subsumption, as opposed to what entity types there are. Hence, one system may have separate representations for managers and engineers, whereas another may model all of the information collectively in an employee "entity type". When translated to relational terms, this means that a single relation in one system may have to be correlated with multiple relations in another. Moreover, some information may now be implied by the structure and hence become implicit as shown in Figure 2-3. This corresponds to the many-to-many table conflicts described by Kim and Seo [1991].

Semantic Heterogeneity

Semantic heterogeneity refers to the fact that data present in different systems may be subjected to different interpretations, even when the corresponding database schemas are identical. A considerable amount has been written about the various types of semantic conflicts, though there appears to be little consensus on what is a good taxonomy. This section organizes these into three distinct categories.

Naming Conflicts

Naming Conflicts consists of synonyms and homonyms among attribute *values*. For example, the name of a company may be reported differently in different systems ("International Business Machines" versus "IBM" versus "I.B.M."). In most instances, these differences are idiosyncratic in that the variations are not systematic. The adoption of standards (even if it means having a multiplicity of them) is often useful in limiting the number of variations down to some small number. In this case, it becomes feasible to have "mapping tables" for translating from one symbolic representation to another. Standards for units of measures is one such example whereby translation from standard names and abbreviations (e.g., "kilometer" versus "km") is routinely done.

Scaling and Units Conflicts

Scaling and Units Conflicts refers to the adoption of different units of measures or scales in reporting. For example, financial data are routinely reported using different currencies and scale-factors. Also, academic grades may be reported on several different scales having different granularities (e.g., a five-point scale with letter grades "A" "B", "C", "D" and "F", or a four-point scale comprising "excellent", "good", "pass", "fail").

Confounding Conflicts

Confounding Conflicts refer to those arising from the confounding of concepts which are in actual fact distinct. For example, the "latest trade price" reported by two data feeds may differ from one another simply because one reports with a greater temporal delay compared to the other. In this instance, the conflict arises from failure to distinguish between the two concepts "latest trade price with a 5-minute delay" versus "latest trade price as of now". Similarly, many variations commonly found in financial statements can be traced to the adoption of different accounting practices. For instance, total assets of a company may vary substantially depending on how asset depreciation is amortized.

Intensional Heterogeneity

Intentional heterogeneity refers to the differences in informational content present in sources or expected by receivers. There are two aspects to this:

Domain Conflicts

Domain Conflicts refer to discrepancies in the *domain*, or the underlying "universe of discourse", which is (implicitly) modeled by each component system. As an illustration, two sources may provide financial information on companies, but the first reports "all US Fortune 500 companies in the manufacturing sector", whereas the second may report information for "all companies listed on US stock exchanges (for-

eign or US-incorporated) with total assets above one billion US Dollars". In general, the extensions of two distinct components may be related in a number of ways: they may be identical, one may be a strict subset of the other, they may be disjoint, or they may overlap in some nontrivial way. Knowing what extensional data sets are provided by disparate sources is important for determining a minimal set of database requests needed to satisfy a given query: if it is known that the extension of source A is a superset of those of sources B and C, querying all three sources is clearly redundant and entails more work than necessary. Receivers too can have implicit extensions. An application may be designed around the assumption that it is operating on a collection qualified in some manner (e.g., all stocks traded on New York Stock Exchange). It is critical that queries issued by this application be understood with reference to the assumptions which bind the extensional data set in order that it can be correctly interpreted.

Integrity Constraint Conflicts

Integrity Constraint Conflicts refer to disparity among the integrity constraints asserted in different systems. The simplest (but potentially most troublesome) form of these is conflicts over *key constraints*. The name of an individual may be unique in one component system (hence allowing it to be used as a key), but not in another. In general, many different possibilities exists given that integrity constraints can take on many different forms. From a data retrieval point of view, certain violations can be treated as inconsistencies and dealt with as such (see, for instance the approach adopted by Agarwal et al. [1995]). Updates under these circumstances are much more difficult and remains a topic for further research.

2.2 Approaches to Achieving Interoperability

Within the last decade or so, there has been a proliferation of proposals and research prototypes aimed at achieving interoperability among autonomous and heterogeneous databases. Primarily, these proposals differ from one another along two dimensions:

- the choice of the underlying data model for achieving schematic and semantic transformations needed for conflict resolution; and
- subscription to either a tight-coupling integration strategy or a loose-coupling integration strategy.

Figure 2-4 identifies some of the better known systems which have been reported in the research literature. Not surprisingly, the semantically-richer data models (i.e., object-oriented and logic-based formalisms) have gained greater popularity over traditional (relational) systems. The distinction between tight- and loose-coupling systems, on the other hand, can be characterized by

- *who* is responsible for identifying what conflicts exists and how they can be circumvented; and
- *when* the conflicts are resolved.

Clearly, the two tasks are correlated since the first must precede the second. The remainder of this section describes the pertinent features of the two approaches. This is followed by a survey of some of the more reported in the literature.

In the discussion which follows, we draw a sharp distinction between tight- and loose-coupling integration strategies to provide a contrast of the tradeoffs between the two approaches. In actual fact, most integration exercises in the real-world fall somewhere in between on this continuum, and few, if any, would venture to the extremes. This "middle-ground" strategy, however, does not seem to offer additional benefits since the two strategies are founded on very different premises on how data semantics are captured and do not leverage effectively on each other. At best, the amalgamation of the two approaches in a real-world scenario provides a mixed bag of interfaces from which receivers can choose from, but does not, in general, circumvent the problems inherent in each strategy.

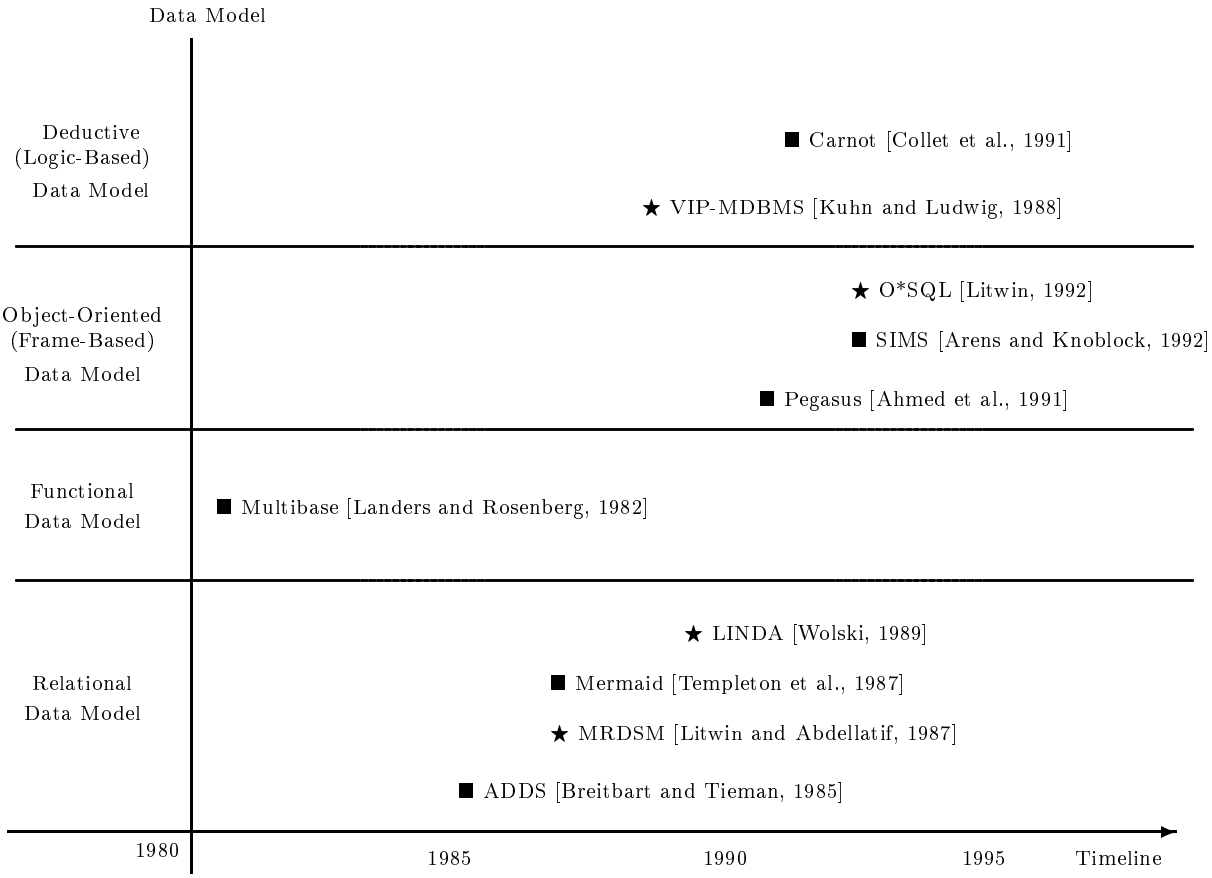

Figure 2-4: A sampling of prototype systems constructed using the tightly-coupled (■) and the loosely-coupled (★) strategies.

The Tight-Coupling Strategy

In the case of tightly-coupled systems, the detection of conflicts (and alternatives for resolving them) is performed by a system administrator and the actual resolution is accomplished by defining, *a priori*, one or more views which define the *shared schemas* for the system. A shared schema insulates the receiver from underlying data heterogeneity by providing a canonical representation of the data originating from disparate sources. Queries formulated against a shared schema can be transformed to subqueries which are submitted to component sources, and the results are translated to the canonical representation using the view definition. Early prototypes which have been constructed using the tight-coupling approach include Multibase [Landers

and Rosenberg, 1982], ADDS [Breitbart and Tieman, 1985], and Mermaid [Templeton et al., 1987]. More recently, the same strategy has been employed for systems adopting object-oriented data models (e.g., Pegasus [Ahmed et al., 1991] based on the IRIS data model), frame-based knowledge representation languages (e.g., SIMS [Arens and Knoblock, 1992] using LOOM), as well as logic-based languages (e.g., Carnot [Collet et al., 1991] using CycL, an extension of first-order predicate calculus).

In virtually all of the above cases, the technique for conflict resolution is similar to that proposed by Dayal and Hwang [1984]: i.e., conflicts in underlying sources are encapsulated via the introduction of a *supertype*, which has methods or functions which are defined with reference to its subtypes. For instance, consider the following conflict for student grades reported by two databases: the first database represents student grades using letter grades, and the second represents the same as points in the range of 0 to 100. In the Pegasus system, this integration is accomplished by introducing a supertype which subsumes the two `Student` types and allowing all attributes of the subtypes to be "upward inherited". Hence, if the attribute `Name` is common to `Student1` and `Student2`, the invocation of method `Name` on `Student` will be automatically translated to the invocation of `Name` on one of the subtypes. Semantic conflicts are circumvented by providing the necessary conversion functions (in this case, `Map1` and `Map2`) to effect the translation to a "canonical" representation:

```
CREATE SUPERTYPE student OF student1, student2;
CREATE FUNCTION score(student x) ->
    REAL r AS
    IF student1(x) THEN map1(grade(x))
    ELSE IF student2(x) THEN map2(points(x))
    ELSE ERROR;
```

As we will point out in the next chapter, there are distinct disadvantages for adopting such a technique for circumventing data conflicts.

The Loose-Coupling Strategy

Systems constructed using the loose-coupling approach, on the other hand, subscribe to the belief that the creation and maintenance of shared schemas is infeasible for any nontrivial number of sources. Hence, instead of resolving *all* conflicts *a priori*, conflict detection and resolution are undertaken by receivers themselves, who need only interact with a limited subset of the sources at any one time. To facilitate this task, research on this front has focused on the invention of powerful data manipulation languages (DMLs) which allow queries on multiple sources to be intermingled with operations for data transformations. MRDSM [Litwin and Abdellatif, 1987], is probably the best-known example of a loosely-coupled system, in which queries are formulated using the multidatabase language MDSL. Kuhn and Ludwig [1988] have implemented similar functionalities in VIP-MDBS, for which queries and data transformations are written in Prolog. They showed that the adoption of a declarative specification does in fact increase the expressiveness of the language in terms of allowable data transformations. More recently, Litwin [1992] has defined another query language called O∗SQL which is largely an object-oriented extension to MDSL.

A novel feature of MDSL is the use of *dynamic attributes*, which take on values returned by conversion operations. Litwin and Abdellatif [1987] have proposed three different operators for accomplishing data transformation: arithmetic operators, mapping tables, and functions in the form of executable code. Returning to the earlier scaling conflict on student grades, a MDSL query for finding all students with an "A"-grade in Database2 (which reports in points in the range 1 to 100) can take the following form:

```
OPEN database2
-RANGE (t student2)
-ATTR_D grade :  CHAR
-DEFINED BY P(score) = map2
-SELECT t -WHERE (t.grade = 'A')
-RETRIEVE
```

In this query, `grade` is a dynamic attribute (as denoted by the keyword *ATTR_D*) which is derived from the actual stored attribute `score` by applying a program (hence the letter `P`) `map2` to it. Notice that the actual select statement is formulated using attribute `grade` as opposed to `score`. As will be demonstrated in the next section, the loose-coupling approach helps to eliminate some of the difficulties inherent in tight-coupling systems but at the same time introduces new problems which renders it less viable than it presumed to be.

2.3 The Measure of a Viable Integration Strategy

Traditionally, the only requirement of an integration strategy is that interoperability must be accomplished in a *non-intrusive* manner: i.e., the provision of new services must not require changes in existing sources nor should it interfere with applications operating on local sources only. Clearly, classical integration approaches described above have more than adequately met this goal.

The recent years, however, have witnessed an exponential growth in the number of sources and receivers which are demanding solutions to complex integration scenarios characterized by a large number of component systems operating in a diversified and dynamic environment[2]. This phenomenon can be attributed to a number of reasons:

- the rise of new organizational forms (e.g., *adhocracies* and *networked organizations* [Malone and Rockart, 1991]) which require information resources to be shared across traditional organizational and functional boundaries;

- advances in telecommunications and networking technology which led to rapidly declining cost/performance ratio, making network investments much more attractive; and

- the development and wide-spread acceptance of standards (e.g., URLs and the

[2]A well-documented example is the Integrated Weapons Systems Data Base (IWSDB) [Wiederhold, 1993], for which more than 50 databases (containing information on technical specifications, design, manufacturing, and operational logistics) have been identified as of 1993, with many more expected over the next five decades.

HTML mark-up language) and network protocols (e.g., the HTTP and IP protocols) which ease the transition to a networked environment.

Whatever the initial reasons may be, these have led to a "snowballing" effect. In economic terms, this is a classic scenario where *positive network externalities* is at play: as more and more organizations make their information resources and applications network-aware, the "marginal benefit" for the next source or receiver to come online becomes greater.

In the light of the preceding observations we suggest that, for any integration strategy to be viable, it must satisfy the following three additional criteria: *scalability*, *extensibility*, and *accessibility*. Unfortunately, this is where classical integration approaches have fallen short.

Scalability

The first measure of a viable integration strategy is undisputedly its ability to scale: i.e., its efficacy must not degrade drastically when the number of component systems increases (say, from three to three hundred). For the problem at hand, it turns out that things are much messier because the number of sources and receivers correlates positively with the extent of data conflicts: i.e., having a larger number of sources and receivers almost certainly means that there will be *quantitatively* more conflicts which are *qualitatively* more diverse. This impacts current integration strategies in at least two ways.

First, the strategy adopted for semantic integration is generally not tenable when the number of sources and receivers is large. In the case of tightly-coupled systems, defining a shared schema encapsulating all of the underlying conflicts becomes prohibitively complex. This is not just because sources are more heterogeneous, but also due to the fact that receivers will most likely have more diverse requirements which must be accounted for in the design of the shared schema. (In some instances, this may require the creation of *several* shared schemas to cater to the needs of different users.) With loosely-coupled systems, receivers, instead of system administrators,

bear the brunt of the work. On the whole, the problem is even more pronounced since end users have limited resources and hence may not be sufficiently equipped to deal with complex semantic issues.

From a query processing point of view, having greater diversity in component systems means that more data conversions are expected leading to the rapid degradation of system performance. To make matters worse, both the tight- and loose-coupling approaches provide little scope for query optimization since semantic conflicts are never represented explicitly but are encapsulated in the form of conversion functions (which cannot be easily inspected or reasoned with by a query optimizer). In a small and controlled environment, it is often possible for "canned" queries to be carefully hand-crafted to meet the needs of critical applications. Such an approach, however, becomes infeasible when a large number of receivers need to be supported and when frequent changes in the underlying system are expected.

Extensibility

Things change over time. In the context of our discussion, changes come in one of two forms: changes in membership of component systems (i.e., when new sources or receivers are added or removed) and changes in schemas and/or semantics of underlying systems. A particularly interesting account of the latter, sometimes referred to as *domain evolution*, has been reported by Ventrone and Heiler [1991] .

Whenever system membership or underlying schemas/semantics change, the effects must somehow be reflected in the rest of the system which interoperates with the element that has been changed. In the case of tightly-coupled systems, this means that shared schemas referencing the altered component must be *manually* updated by a system administrator. With loosely-coupled systems, such changes must be conveyed to receivers, who have vested interest in them, who must somehow register the changes to facilitate future access. Individually, the changes may occur infrequently. However, a small number of infrequent changes at the component level can add up to formidable recurring events at the system level. For example, assuming an average of one change in three years for any component system, an integrated system

with three hundred sources will have to contend with a hundred changes every year, translating to two changes every week! The reliance of classical strategies on manual interventions in mitigating these changes is clearly not a feasible choice.

Accessibility

The third criteria which we deem to be important is the ability to make information resources more accessible to receivers. There are at least two variations of this theme.

First, given the proliferation of sources and their inherent heterogeneity, figuring out "what relevant information exists and where they can be found" is increasingly a problem in itself [Bouguettaya and King, 1992]. This is sometimes referred to in the literature as the *resource discovery* problem. Despite the criticisms leveled at the tight-coupling systems, it has remained a better strategy compared to loosely-coupled system *in this regard*. However, the integration of new sources often entails changes to the shared schema which cause the latter to be a bottleneck.

Second, accessibility also means being able to present information to a receiver in the structure or representation that the latter expects. We have alluded to this earlier by pointing out that receivers, like sources, are heterogeneous and any integration strategy must seek to preserve their autonomy: i.e., receivers should be able to issue the same queries and receive answers in the same format/interpretation as they would normally expect (say, from a local data source). Traditionally, this is accomplished in tightly-coupled systems by allowing receivers to define one or more *external views* which are used as the basis for query formulation. This approach suffers from the problems mentioned earlier concerning the creation and maintenance of shared schemas.

Every general theory must work in at least one case.

– Stuart E. Madnick

Chapter 3

Context Interchange By Example

The goal of this chapter is to provide a high-level tour of the Context Interchange approach which will highlight various features of the proposed strategy through the use of examples. Throughout this discussion, we make the assumption that the relational data model is adopted to be the *canonical data model* [Sheth and Larson, 1990]: i.e., we assume that the database schemas exported by the sources are relational and that queries are formulated using SQL (or some extension thereof). This simplifies the discussion by allowing us to focus on semantic conflicts in disparate systems without being detracted by conflicts over data model constructs. The choice of the relational data model is one of convenience rather than necessity, and is not to be construed as a constraint of the integration strategy being proposed.

We present the scenario underlying the example in the next section. Discussion concerning the features of Context Interchange is organized in two parts: the first examines the distinctive properties of our approach from a user perspective; the second focuses on features that are novel from a system perspective, with particular attention to its scalability and extensibility. The final section contrasts our integration strategy to classical and contemporary integration approaches to provide a better appreciation of these features.

3.1 Scenario Description

Consider the scenario shown in Figure 3-1, deliberately kept simple for didactical reasons. Data on "revenue" and "expenses" (respectively) for some collection of companies are available in two autonomously-administered data sources, each comprised of a single relation Suppose a user is interested in knowing which companies have been "profitable" and their respective revenue: this query can be formulated directly on the (export) schemas of the two sources as follows[1]:

Q1: `SELECT r1.cname, r1.revenue FROM r1, r2`
`WHERE r1.cname = r2.cname AND r1.revenue > r2.expenses;`

In the absence of any mediation, this query will return the empty answer if it is executed over the extensional data set shown in Figure 3-1.

The above query, however, does not take into account the fact that data sources are administered independently and have different *contexts*: i.e., they may embody different assumptions on how information contained therein should be interpreted. To simplify the ensuing discussion, we assume that the data reported in the two sources differ only in the currencies and scale-factors of "company financials" (i.e., financial figures pertaining to the companies, which include `revenue` and `expenses`). Specifically, in Source 1, all "company financials" are reported using the currency shown and a scale-factor of 1; the only exception is when they are reported in Japanese Yen (`JPY`); in which case the scale-factor is `1000`. Source 2, on the other hand, reports all "company financials" in `USD` using a scale-factor of `1`. In the light of these remarks, the (empty) answer returned by executing Q1 is clearly not a "correct" answer since the revenue of `NTT` (`9,600,000 USD` = `1,000,000` × `1,000` × `0.0096`) is numerically larger than the expenses (`5,000,000`) reported in `r2`.

[1] We assume, without loss of generality, that relation names are unique across all data sources. This can always be accomplished via some renaming scheme: say, by prefixing relation name with the name of the data source (e.g., `db1#r1`).

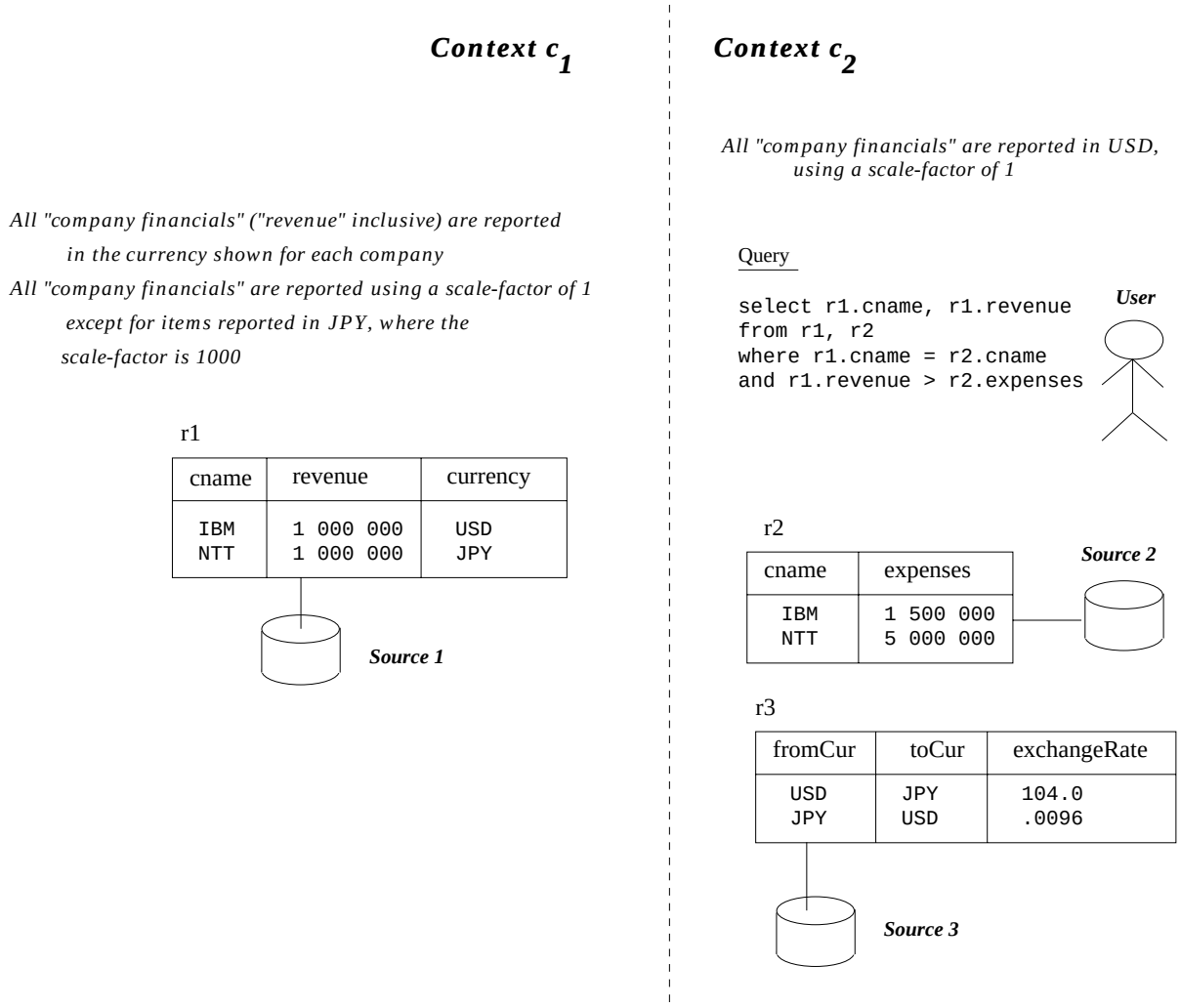

Figure 3-1: Scenario for the motivational example.

3.2 A User Perspective of Context Interchange

Unlike classical and contemporary approaches, the Context Interchange approach provides users with a wide array of options on *how* and *what queries* can be asked and the *kinds of answers* which can be returned. These features work in tandem to allow greater flexibility and effectiveness in gaining access to information present in multiple, heterogeneous systems.

Query Mediation: Automatic Detection and Reconciliation of Conflicts

In a Context Interchange system, the same query (Q1) can be submitted to a specialized *mediator* [Wiederhold, 1992], called a *Context Mediator*, which rewrites the query so that data exchange among sites having disparate contexts are interleaved with appropriate data transformations and access to ancillary data sources (when needed). We refer to this transformation as *query mediation* and the resulting query as the corresponding *mediated query*.

For example, the mediated query MQ1 corresponding to Q1 is given by:

MQ1:
```
SELECT r1.cname, r1.revenue FROM r1, r2
WHERE r1.currency = 'USD' AND  r1.cname = r2.cname
AND r1.revenue > r2.expenses;
UNION
SELECT r1.cname, r1.revenue * 1000 * r3.rate FROM r1, r2, r3
WHERE r1.currency = 'JPY' AND r1.cname = r2.cname
AND r3.fromCur = r1.currency AND r3.toCur = 'USD'
AND r1.revenue * 1000 * r3.rate > r2.expenses
UNION
SELECT r1.cname, r1.revenue * r3.rate FROM r1, r2, r3
WHERE r1.currency <> 'USD' AND  r1.currency <> 'JPY'
AND r3.fromCur = r1.currency AND r3.toCur = 'USD'
AND r1.cname = r2.cname AND r1.revenue * r3.rate > r2.expenses;
```

This mediated query considers all potential conflicts between relations `r1` and `r2` when comparing values of "revenue" and "expenses" as reported in the two different contexts. Moreover, the answers returned may be further transformed so that they conform to the context of the receiver. Thus in our example, the revenue of `NTT` will be reported as `9 600 000` as opposed to `1 000 000`. More specifically, the three-part query shown above can be understood as follows. The first subquery takes care of tuples for which revenue is reported in `USD` using scale-factor 1; in this case, there is

no conflict. The second subquery handles tuples for which revenue is reported in `JPY`, implying a scale-factor of `1000`. Finally, the last subquery considers the case where the currency is neither `JPY` nor `USD`, in which case only currency conversion is needed. Conversion among different currencies is aided by the ancillary data source `r3` which provides currency conversion rates. This second query, when executed, returns the "correct" answer consisting only of the tuple <'NTT', 9 600 000>.

Support for Views

In the preceding example, the query Q1 is formulated directly on the export schema for the various sources. While this provides a great deal of flexibility, it also requires users to know what data are present *where* and be sufficiently familiar with the attributes in different schemas (so as to construct a query). A simple and yet effective solution to these problems is to allow *views* to be defined on the source schemas and have users formulate queries based on the view instead. For example, we might define a view on relations `r1` and `r2`, given by

```
CREATE VIEW v1 (cname, profit) AS
SELECT r1.cname, r1.revenue - r2.expenses
FROM r1, r2
WHERE r1.cname = r2.cname;
```

In which case, query Q1 can be equivalently formulated on the view `v1` as

```
VQ1:    SELECT cname, profit FROM v1
        WHERE profit > 0;
```

While achieving essentially the same functionalities as tightly-coupled systems, notice that view definitions in our case are no longer concerned with semantic heterogeneity and make no attempts at identifying or resolving conflicts. In fact, any query on a view (say, VQ1 on `v1`) can be trivially rewritten to a query on the source schema (e.g., Q1). This means that query mediation can be undertaken by the Context Mediator as before.

Knowledge-Level versus Data-Level Queries

Instead of inquiring about stored data, it is sometimes useful to be able to query the semantics of data which are implicit in different systems. Consider, for instance, the query based on a superset of SQL[2]:

```
Q2:     SELECT r1.cname, r1.revenue.scaleFactor IN c1,
               r1.revenue.scaleFactor IN c2 FROM r1
        WHERE r1.revenue.scaleFactor IN c1 <>
               r1.revenue.scaleFactor IN c2;
```

Intuitively, this query asks for companies for which scale-factors for reporting "revenue" in `r1` (in context c1) differ from that which the user assumes (in context c2). We refer to queries such as Q2 as *knowledge-level queries*, as opposed to *data-level queries* which are enquires on factual data present in data sources. Knowledge-level queries have received little attention in the database literature and certainly have not been addressed by the data integration community. This, in our opinion, is a significant gap since heterogeneity in disparate data sources arises primarily from incompatible assumptions about how data are interpreted. Our ability to integrate access to both data and semantics can be exploited by users to gain insights into differences among particular systems ("Do sources A and B report a piece of data differently? If so, how?"), or by a query optimizer which may want to identify sites with minimal conflicting interpretations (to minimize costs associated with data transformations).

Interestingly, knowledge-level queries can be answered using the exact same inference mechanism for mediating data-level queries. Hence, submitting query Q2 to the Context Mediator will yield the result:

```
MQ2:    SELECT r1.cname, 1000, 1 FROM r1
        WHERE r1.currency = 'JPY';
```

[2]Sciore et al. [Sciore et al., 1992] have described a similar (but not identical) extension of SQL in which context is treated as a "first-class object". We are not concern with the exact syntax of such a language here; the issue at hand is how we might support the underlying inferences needed to answer such queries.

which indicates that the answer consists of companies for which the currency attribute has value `'JPY'`, in which case the scale-factors in context c1 and c2 are `1000` and `1` respectively. If desired, the mediated query MQ2 can be evaluated on the extensional data set to return an answer grounded in actual data elements. Hence, if MQ2 is evaluated on the data set shown in Figure 3-1, we would obtain the singleton answer <`'NTT'`, `1000`, `1`>.

Extensional versus Intensional Answers

Yet another feature of Context Interchange is that *answers* to queries can be both intensional and extensional. Extensional answers correspond to fact-sets which one normally expects of a database retrieval. Intensional answers, on the other hand, provide only a characterization of the extensional answers *without* actually retrieving data from the data sources. In the preceding example, MQ2 can in fact be understood as an intensional answer for Q2, while the tuple obtained by the evaluation of MQ2 constitutes the extensional answer for Q2. In the COIN framework, intensional answers are grounded in extensional predicates (i.e., names of relations), evaluable predicates (e.g., arithmetic operators or "relational" operators), and external functions which can be directly evaluated through system calls. The intensional answer is thus no different from a query which can normally be evaluated on a conventional query subsystem of a DBMS. Query answering in a Context Interchange system is thus a two-step process: an intensional answer is first returned in response to a user query; this can then be executed on a conventional query subsystem to obtain the extensional answer.

The intermediary intensional answer serves a number of purposes [Imielinski, 1987]. Conceptually, it constitutes the mediated query corresponding to the user query and can be used to confirm the user's understanding of what the query actually entails. More often than not, the intensional answer can be more informative and easier to comprehend compared to the extensional answer it derives. (For example, the intensional answer MQ2 actually conveys more information than merely returning the single tuple satisfying the query.) From an operational standpoint, the

computation of extensional answers are likely to be many orders of magnitude more expensive compared to the evaluation of the corresponding intensional answer. It therefore makes good sense not to continue with query evaluation if the intensional answer satisfies the user. From a practical standpoint, this two-stage process allows us to separate query mediation from query optimization and execution. As we will illustrate later in this paper, query mediation is driven by logical inferences which do not bond well with the (predominantly cost-based) optimization techniques that have been developed [Mumick and Pirahesh, 1994, Seshadri et al., 1996]. The advantage of keeping the two tasks apart is thus not merely a conceptual convenience, but allows us to take advantage of mature techniques for query optimization in determining how best a query can be evaluated.

Query Pruning

Finally, observe that consistency checking is performed as an integral activity of the mediation process, allowing intensional answers to be pruned (in some cases, significantly) to arrive at answers which are better comprehensible and more efficient. For example, if Q1 had been modified to include the additional condition "`r1.currency = 'JPY'`", the intensional answer returned (MQ1) would have only the second `SELECT` statement (but *not* the first and the third) since the other two would have been inconsistent with the newly imposed condition. This pruning of the intensional answer, accomplished by taking into consideration integrity constraints (present as part of a query, or those defined on sources) and knowledge of data semantics in distinct systems, constitutes a form of *semantic query optimization* [Chakravarthy et al., 1990]. Consistency checking however can be an expensive operation and the gains from a more efficient execution must be balanced against the cost of performing the consistency check during query mediation. In our case, however, the benefits are amplified since spurious conflicts that remain undetected could result in an additional conjunctive query involving multiple sources.

3.3 A System Perspective of Context Interchange

It is natural to assume the internal complexity of any system will increase in commensuration with the external functionalities it offers. The Context Interchange system is no exception. We make no claim that our approach is "simple"; however, we submit that this complexity is decomposable and well-founded. *Decomposability* has obvious benefits from a system engineering perspective, allowing complexity to be harnessed into small chunks, thus making our integration approach more endurable, even when the number of sources and receivers are exponentially large and when changes are rampant. The complexity is said to be *well-founded* because it is possible to characterize the behavior of the system in an abstract mathematical framework. This allows us to understand the potential (or limits) of the strategy apart from the idiosyncrasies of the implementation, and is useful for providing insights into where and how improvements can be made. We describe below some of the ways in which complexity is decoupled in a Context Interchange system, as well as tangible benefits which result from formalization of the integration strategy.

Representation of 'Meaning' as opposed to Conflicts

As mentioned earlier, a key insight of the Context Interchange strategy is that we can represent the meaning of data in the underlying sources and receivers *without* identifying and reconciling all potential conflicts which exist between any two systems. Thus, query mediation can be performed dynamically (as when a query is submitted) or it can be used to produce a query plan (the mediated query) that constitutes a locally-defined view. In the latter case, this view is similar to the shared schemas in tightly-coupled systems with one important exception: whenever changes do occur (say, when the semantics of data encoded in some context is changed[3]), the Context Mediator can be triggered to reconstruct the local view automatically. Unlike the case in tightly-coupled systems, this reconstruction requires no manual intervention

[3]For an account of why this seemingly strange phenomenon may be more common than is widely believed to be, see [Ventrone and Heiler, 1991].

from the system administrators. In both of the above scenarios, changes in local systems are well-contained and do not mandate human intervention in other parts of the larger system. This represents a significant gain over tightly-coupled systems where maintenance of shared schemas constitute a major system bottleneck.

Contexts versus Schemas

Unlike the semantic-value model, the COIN data model adopts a very different conceptualization of *contexts*: instead of a property-list associated with individual data elements, we view context as consisting of a collection of axioms which describes a particular "situation" a source or receiver is in.

Figure 3-2 provides a graphical representation of the salient features of the structure which is the key enabler for this representation. The *domain model* presents the definitions for the "types" of information units (called *semantic-types*) that constitute a common vocabulary for capturing the semantics of data in disparate systems. Instances of semantic-types are called *semantic-objects*. Every data element in a source or receiver is mapped to a unique semantic-object for which the object-id is a *Skolem function* [Chang and Lee, 1973] defined on some key values. For each relation r present in a source, there is an isomorphic relation r' defined on the corresponding semantic-objects. Among other features, this structure allows us to encode assumptions of the underlying context independently of structure imposed on data in underlying sources (i.e., the schemas). For example, the constraint: all "company financials" are reported in US Dollars within context c_2 can be described using the clause[4]

$$X'{:}companyFinancials,\ currency(c_2, X'){:}number \vdash$$
$$currency(c_2, X')[value(c_2) \rightarrow \texttt{'USD'}].$$

A detailed presentation of the details of this framework will be presented later in Chapter 4.

[4]The syntax of this language is defined in Chapter 4 and corresponds to that of Gulog [Dobbie and Topor, 1995].

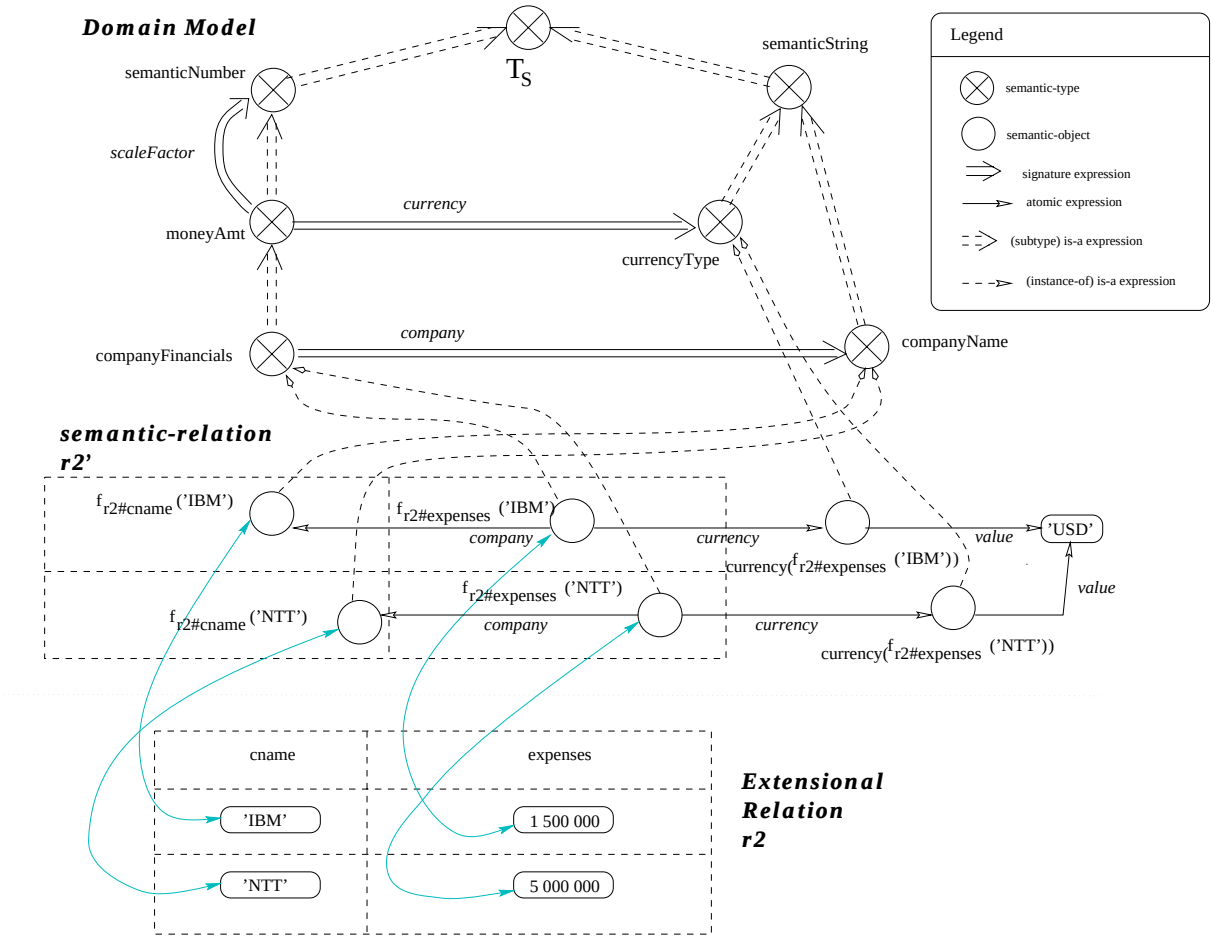

Figure 3-2: A graphical representation of the relationships between semantic-types in the domain model, semantic-relations (defined on semantic-objects), and data elements in the relation r2.

The dichotomy between schemas and contexts present a number of opportunities for systematic sharing and reuse of semantic encodings. For example, different sources and receivers in the same context may now bind to the same set of context axioms; and distinct attributes which correlate with one another (e.g., `revenue`, `expenses`) may be mapped to instances of the same semantic-type (e.g., *companyFinancials*). These circumvent the need to define a new property-list for all attributes in each schema. Unlike the semantic value model, there is no ambiguity on what "labels" can be introduced as "meta-attributes" since all properties of semantic-types are explicitly defined in the domain model. As pointed out in the previous section, views

can be (more simply) defined on the extensional relation independently of the context descriptions. This constitutes yet another benefit of teasing apart the structure of a source, and semantics which are implicit in its context.

Inheritance and Overriding in Semantic-Types

Not surprisingly, the various features frequently associated with "object-orientation" are useful in our representation scheme as well. Semantic-types fall naturally into a generalization hierarchy, which allow us to take advantage of structural and behavioral inheritance in achieving economy of expression. *Structural inheritance* allows a semantic-type to inherit the declarations defined for its supertypes. For example, the semantic-type *companyFinancials* inherits from *moneyAmt* the declarations concerning the *existence* of the "methods" *currency* and *scaleFactor*. *Behavioral inheritance* allows the *definitions* of these methods to be inherited as well. Hence, if we had defined earlier that instances of *moneyAmt* has a scale-factor of `1`, all instances of *companyFinancials* would inherit the same scale-factor since every instance of *companyFinancials* is an instance of *moneyAmt*.

Inheritance need not be monotonic. Non-monotonic inheritance means that the declaration or definition of a method can be overridden in a subtype. Thus, inherited definitions can be viewed as *defaults* which can always be changed to reflect the specificities at hand.

Value Conversion Among Different Contexts

Yet another benefit of adopting an object-oriented model in our framework is that it allows conversion functions on values to be explicitly defined in the form of methods defined on various semantic types. For example, the conversion function for converting an instance of *moneyAmt* from one scale-factor (F in context C) to another (F_1 in context c_1) can be defined as follows:

$$X' : moneyAmt \vdash$$
$$X'[cvt(c_1)@scaleFactor, C, U \rightarrow V] \leftarrow$$

$$X'[scaleFactor(c_1) \rightarrow \_[value(c_1) \rightarrow F_1]],$$
$$X'[scaleFactor(C) \rightarrow \_[value(c_1) \rightarrow F]], V = U * F/F_1.$$

This conversion function, unless explicitly overridden, will be invoked whenever there is a request for scale-factor conversion on an object which is an instance of *moneyAmt* and when the conversion is to be performed with reference to context c_1. Overriding can take place along the generalization hierarchy: as before, we may introduce a different conversion function for a subtype of *moneyAmt*. Notice that this conversion function is defined with reference to context c_1 only: in order for scale-factor conversion to take place in a different context, the conversion function (which could be identical to the one in c_1, or not) will have to be defined explicitly. This phenomenon allows different conversion functions to be associated with different contexts and is a powerful mechanism for different users to introduce their own interpretations of disparate data in a localized fashion. The apparent redundancy (in having multiple instances of the same definition in different contexts) is addressed through the adoption of a context hierarchy which is described next.

Hierarchical composition of Contexts

By "objectifying" sets of axioms associated with contexts, we can introduce a hierarchical relationship among contexts. If c is a *subcontext* of c', then all the axioms defined in c' are said to apply in c unless they are "overridden". An immediate application of this concept is to make all "functional" contexts subcontexts of a *default context* c_0, which contains the default declarations and method definitions. Under this scheme, new contexts introduced need only to identify how it is different from the default context and introduce the declarations and method definitions which need to be changed (overridden). This is formulated as a meta-logical extension of the COIN framework and will be described in further details in Section 4.6.

Another advantage of having this hierarchy of context is the ability to introduce changes to the domain model in an incremental fashion without having adverse effects on existing systems. For example, suppose we need to add a new source for which cur-

rency units take on a different representation (e.g., `'Japanese Yen'` versus `'JPY'`). This distinction has not been previously captured in our domain model, which has hitherto assumed currency units have a homogeneous representation. To accommodate the new data source, it is necessary to add a new modifier for *currencyType*, say *format*, in the domain model:

$$currencyType[format(ctx) \Rightarrow semanticString].$$

Rather than making changes to all existing contexts, we can assign a default value to this modifier in c_0, and at the same time, introduce a conversion function for mapping between currency representations of different formats (e.g., `'Japanese Yen'` and `'JPY'`):

$$X{:}currencyType,\ format(c_0, X){:}semanticString \vdash$$
$$format(c_0, X)[value(c_0) \rightarrow \texttt{'abbreviated'}].$$
$$X{:}currencyType \vdash X[cvt(c_0)@C, U{\rightarrow}V] \leftarrow \ldots (\text{body}) \ldots$$

The last step in this process is to add to the new context (c') the following context axiom:

$$X{:}currencyType, format(c', X){:}semanticString \vdash$$
$$format(c', X)[value(c') \rightarrow \texttt{'full'}].$$

which distinguishes it as having a different *format*.

3.4 Context Interchange vis-à-vis Traditional and Contemporary Integration Approaches

In the preceding sections, we have made detailed comments on the many features that the Context Interchange approach has over traditional loose- and tight-coupling approaches. In summary, although tightly-coupled systems may provide better support for data access to heterogeneous systems (compared to loosely-coupled systems), they do not scale-up effectively given the complexity involved in constructing a shared

schema for a large number of systems and are generally unresponsive to changes for the same reason. Loosely-coupled systems, on the other hand, require little central administration but are equally non-viable since they require users to have intimate knowledge of the data sources being accessed; this assumption is generally non-tenable when the number of systems involved is large and when changes are frequent[5]. The Context Interchange approach provides a novel middle ground between the two: it allows queries to sources to be mediated in a transparent manner, provides systematic support for elucidating the semantics of data in disparate sources and receivers, and at the same time, does not succumb to the complexities inherent in maintenance of shared schemas.

At a cursory level, the Context Interchange approach may appear similar to many contemporary integration approaches. Examples of these commonalities include:

- framing the problem in McCarthy's *theory of contexts* [McCarthy, 1987] (as in Carnot [Collet et al., 1991], and more recently, [Faquhar et al., 1995]);
- *encapsulation* [Atkinson et al., 1989] of semantic knowledge in a hierarchy of *rich data types* which are refined via *sub-typing* (as in several object-oriented multidatabase systems, the archetype of which is Pegasus [Ahmed et al., 1991]);
- adoption of a *deductive* or *object-oriented formalism* [Kifer et al., 1995, Dobbie and Topor, 1995] (as in the ECRC Multidatabase System [Jonker and Schütz, 1995] and DISCO [Tomasic et al., 1995]);
- provision of value-added services through the use of *mediators* [Wiederhold, 1992] (as in TSIMMIS [Garcia-Molina et al., 1995]);

We posit that despite these superficial similarities, our approach represents a radical departure from these contemporary integration strategies.

[5]We have drawn a sharp distinction between the two here to provide a contrast of their relative features. In practice, one is most likely to encounter a hybrid of the two strategies. It should however be noted that the two strategies are incongruent in their outlook and are *not* able to easily take advantage of each other's resources. For instance, data semantics encapsulated in a shared schema cannot be easily extracted by a user to assist in formulating a query which seeks to reference the source schemas directly.

To begin with, a number of contemporary integration approaches are in fact attempts aimed at rejuvenating the loose- or tight-coupling approach. These are often characterized by the adoption of an object-oriented formalism which provides support for more effective data transformation (e.g., O*SQL [Litwin, 1992]) or to mitigate the effects of complexity in schema creation and change management through the use of abstraction and encapsulation mechanisms. To some extent, contemporary approaches such as Pegasus [Ahmed et al., 1991], the ECRC Multidatabase Project [Jonker and Schütz, 1995], and DISCO [Tomasic et al., 1995] can be seen as examples of the latter strategy. These differ from the Context Interchange strategy since they continue to rely on human intervention in reconciling conflicts a priori and in the maintenance of shared schemas. Yet another difference is that although a deductive object-oriented formalism is also used in the Context Interchange approach, "semantic-objects" in our case exist only conceptually and are never actually materialized. One implication of this is that mediated queries obtained from the Context Mediator can be further optimized using traditional query optimizers or be executed by the query subsystem of classical (relational) query subsystems without changes.

In the Carnot system [Collet et al., 1991], semantic interoperability is accomplished by writing *articulation axioms* which translate "statements" which are true in individual sources to statements which are meaningful in the Cyc knowledge base [Lenat and Guha, 1989]. A similar approach is adopted in [Faquhar et al., 1995], where it is suggested that domain-specific *ontologies* [Gruber, 1991], which may provide additional leverage by allowing the ontologies to be shared and reused, can be used in place of Cyc. While we like the explicit treatment of contexts in these efforts and share their concern for sustaining an infrastructure for data integration, our realization of these differ significantly. First, *lifting axioms* [Guha, 1991] in our case operate at a finer level of granularity: rather than writing axioms which map "statements" present in a data source to a common knowledge base, they are used for describing "properties" of individual "data objects". Second, instead of having an "ontology" which captures all structural relationships among data objects (much like a "global schema"), we have a domain model which is a much less elaborate collection

of complex *semantic-types*. These differences account largely for the scalability and extensibility of our approach.

Finally, we remark that the TSIMMIS [Papakonstantinou et al., 1995, Quass et al., 1995] approach stems from the premise that information integration could not, and should not, be fully automated. With this in mind, TSIMMIS opted in favor of providing both a framework and a collection of tools to assist humans in their information processing and integration activities. This motivated the invention of a "light-weight" object model which is intended to be *self-describing*. For practical purposes, this translates to the strategy of making sure that attribute labels are as descriptive as possible and opting for free-text descriptions ("man-pages") which provide elaborations on the semantics of information encapsulated in each object. We concur that this approach may be effective when the data sources are ill-structured and when consensus on a shared vocabulary cannot be achieved. However, there are also many other situations (e.g., where data sources are relatively well-structured and where *some* consensus can be reached) where human intervention is not appropriate or necessary: this distinction is primarily responsible for the different approaches taken in TSIMMIS and our strategy.

Everything should be made as simple
as possible, but no simpler.
– Albert Einstein

Chapter 4

The COIN Data Model

Our goal in this chapter is to provide a formal definition of the COIN data model. As pointed out in [Ullman, 1991a], a *data model* consists of two parts:

- a notation for describing data, and
- a set of operations used to manipulate that data.

The COIN data model is a "logical" data model in the sense that it uses mathematical logic as a way for representing knowledge (of the data being modeled) and as a language for expressing operations on those knowledge structures. At the same time, it is an "object-oriented" data model because it adopts an "object-centric" view of the world and supports many of the features (e.g., object-identity, type hierarchy, inheritance, and overriding) commonly associated with "object-orientation".

Following this introduction, we lay the groundwork for subsequent discussion with with a summary of key concepts found in the literature on deductive data models, object-oriented data models, and deductive object-oriented data models. Section 4.2 introduces the notion of context according to McCarthy [McCarthy, 1987]. Section 4.3 describes the structural elements of the COIN data model. The syntax and informal semantics of the language is presented in Section 4.4.

4.1 Background

Object-oriented data models are seen as natural extensions of *semantic data models* [Hull and King, 1987, Peckham and Maryanski, 1988] providing an object-centric modeling formalism, and showing immense promises in overcoming the *impedance mismatch* between general-purpose programming languages and traditional data manipulation languages (DML). Unfortunately, "object-orientation" became synonymous with a laundry-list of features and few agreements have been reached on which of those truly characterize an object-oriented data model (see for instance, [Atkinson et al., 1989]). *Deductive data models*, on the other hand, are founded on a firm logical formalism that has great appeal to many researchers; its main caveat is that it maintains a "relational" viewpoint and provides little or no *abstraction mechanisms* [Smith and Smith, 1977]. The introduction of *deductive object-oriented data models* is largely a response to the inadequacies its precursors, aimed at providing the features of both data models while circumventing their problems. In this section, we summarize briefly the key features of these data models mentioned above. It is not our intent to provide an in-depth treatment of these subjects; our goal, instead, is to provide an overview of the pertinent concepts and issues that provide the background for subsequent material.

Object-Oriented Data Models

In *object-oriented data models* [Zdonik and Maier, 1990], the "universe of discourse" is captured in the form of a collection of object templates called *types* or *classes*. Every real-world entity is represented by an *object*, which is an instance of some type. The type corresponding to an object defines a template for its structure, which includes both the data-structure and the behavioral aspects of that object. Each object is identified by a unique *object identifier (oid)*, which serves as the basis for sharing and the construction of *complex objects*. Every object has a state defined via a set of *properties*, which can be queried or operated upon through the use of *methods*. This feature is referred to as *encapsulation*.

Methods can be *functional* (i.e., single-valued) or *multi-valued*[1]. A functional method returns only one value when the method is applied to an object; a multi-valued method returns one or more values. The definition of a method has two components: *signature* and *body*. The signature specifies the name of the method, the number and types of arguments it expects, and the type(s) of the result. The body represents the implementation of the method, which can be realized by a sequence of instructions written in some programming languages.

In addition to the notion of "encapsulation", an important feature of object-oriented systems is the ability to capture relationships between different types in the form of a *generalization hierarchy*. A generalization hierarchy allows types to be arranged in some partial order related through the *subtype* relationship. Hence, if t and t' are types and t is a subtype of t', we say that t' is a *supertype* of t; moreover, the subtype t *inherits* the methods defined for the supertype t' and may have additional methods of its own.

It is often useful to distinguish between two types of inheritance. If a method defined in a supertype is redefined by a subtype, we say that the method is being *overridden*. In this case, inheritance is said to be *non-monotonic*; otherwise, it is *monotonic*. Examples of non-monotonic inheritance and overriding have been presented earlier in Chapter 3 in the context of the motivational example.

Yet another source of conflict can be attributed to *multiple inheritance*, where a type inherits directly from more than one parent. For example, *teachingAssistant* may inherit from both *staff* and *student*. A conflict occurs when the supertypes involved have different implementations for a method in common. A number of different strategies have been proposed for overcoming this problem; for example, the conflict may be prohibited (as in C++ [Stroutstrup, 1986]), the corresponding supertypes may be statically ordered to determine a particular sequence of inheritance (as in CLOS [Moon, 1989]), or the user may be compelled to specify which method implementation is preferred (as in O_2 [Bancilhon et al., 1992]).

[1]They could also be "procedural", in which case the methods are used for achieving some side-effects and do not return any meaningful value.

Deductive Data Models

In *deductive data models* [Ceri et al., 1990], a *first-order language* [Lloyd, 1987] is used uniformly to describe both underlying data structures as well as operations which are allowed on these structures. Rather than making inferences in full first-order predicate calculus, it is customary to restrict the underlying language to sublanguages which are computationally less expensive. The simplest of these, called *Datalog*, admit only *Horn clauses*, which are statements of the form:

$$L_0 \leftarrow L_1, \ldots, L_n, \quad n \geq 0$$

where each L_i is a *literal* $p(t_1, \ldots, t_n)$, where p_i is a *predicate* symbol and t_i are *terms*. L_0 is said to be the *head* and $L_1, \ldots, L_n$ the *body* of the statement. Clauses with an empty body are called facts (i.e., the literal in the head is unconditionally true); clauses with at least one literal in the body are called *rules*. In the deductive database literature, it is commonly assumed that the fact set is physically stored in a relational database. The collection of facts in the database is often referred to as the *extensional database (EDB)*; the corresponding rule set is referred to as the *intensional database (IDB)*.

From a logic-programming perspective, the collection of clauses is said to constitute a *program*. The *semantics* of a Datalog program can be defined in three ways [Ullman, 1991a]. The *proof-theoretic* interpretation of a program consists of the set of facts which can be derived (using the inference rules of classical first-order proof theory) from the EDB and the IDB. The *model-theoretic* interpretation considers the meaning of the program as given by the *minimal model* with respect to a given *interpretation* (i.e., an assignment of ground terms in the language to objects in an underlying domain, and the assignment of a boolean value to every possible instance of all predicates) [Lloyd, 1987]. Finally, the *computational* meaning of a program can be defined by providing an algorithm for "executing" the program to decide whether a potential fact is true or false. For pure Datalog, all the three interpretations describe above coincide with one another.

Pure Datalog as described above is very restrictive and consequently has motivated a number of extensions. We consider here Datalog extended with built-in predicates and function symbols [Ullman, 1991b].

Built-in predicates are denoted by special predicate symbols such as $<, >, =, \neq$ with a predefined meaning and are allowed to occur only in the rule body. In addition to the *relational predicates* above, built-in predicates could also be *arithmetic*. For example, the predicate $plus(X, Y, Z)$ can be used to denote the arithmetic expression $X + Y = Z$. For most purposes, built-in predicates can be treated as ordinary EDB-predicates; the only difference being that they correspond to infinite relations and are implemented as procedures which are evaluated during execution (as opposed to being stored in the extensional database). For this reason, extra care is required to make sure that the corresponding Datalog program is *safe* (i.e., that only a finite set of answers are derived). Fortunately, this safety condition can be easily verified using syntactic criteria (see [Ullman, 1991a] for details).

Unlike built-in predicates, *function symbols* are "uninterpreted" symbols (i.e., the "interpretation" or meaning is independently assigned by the user). With the admission of function symbols into the language, a *term* can either be a constant symbol, a variable symbol, or the token $f(t_1, \ldots, t_n)$ where f is a function symbol and $t_1, \ldots, t_n$ are terms. This extension allows for an infinite number of terms to be introduced, and provides the means for modeling *complex objects*. For instance, we could make reference to a person "John Doe" using the term *person*(*lastname*("Doe"), *firstname*("John")). The caveat is that unification becomes more expensive and additional syntactic restrictions have to be observed to guarantee safety (as is the case for built-in predicates).

The literals in the body of a Datalog statement are all required to be *positive*. Statements of the form

$$A \leftarrow B_1, \ldots, B_n$$

where B_i can be an atom $p(\vec{X})$ or the negation of an atom $\neg p(\vec{X})$ is called a *normal Horn clause*, and language which admits these statements are sometimes referred

to as *Datalog with negation*. From a semantic standpoint, negation is incorporated into Datalog through the adoption of the *Closed World Assumption (CWA)* [Reiter, 1978], which allows us to conclude that the negation of a fact is true with respect to a *program* P (i.e., a set of clauses) if it is not a logical consequence of P. The extension of pure Datalog to deal with negative information has been impeded by the fact that there could be several (incompatible) minimal models for a given program whenever negation is present. This problem can be overcomed by making certain syntactic restrictions on such programs; specifically, it has been shown that whenever negation is *stratified*, then it has a unique *perfect model* which arguably provides the correct intended semantics [Przymusinski, 1987]. This is extended in [Van Gelder et al., 1988] which introduces the notion of *well-founded semantics* which deals with arbitrary (i.e., not necessarily stratified) logic programs with negation.

A major focus of deductive database systems is the study of efficient methods for query evaluation. A Datalog query can be evaluated in two ways: *bottom-up*, starting from existing facts and inferring new facts using rules defined in the IDB, or *top-down*, where a goal is recursively transformed to a series of subgoals which terminates in a fact. Within this broad framework, a variety of different methods for query evaluation and rewriting have been proposed. An in-depth description of each of these proposals and a comparison of their relative efficiency can be found in [Bancilhon and Ramakrishnan, 1986].

Deductive Object-Oriented Data Models

As noted earlier, the interest behind *deductive object-oriented data models* is largely a response to inadequacies of object-oriented data models and deductive data models. Despite this common endeaviour, these efforts vary somewhat depending on the research traditions in which they are grounded. For example, both LOGIN [Aït-Kaci and Nasr, 1986] and *L&O* [McCabe, 1992] are in fact object-oriented extensions to classical *logic programming languages* [Lloyd, 1987]. On the other hand, proposals such as *O-logic* [Maier, 1986, Kifer and Wu, 1989], *C-logic* [Chen and Warren, 1989], *F-logic* [Kifer and Lausen, 1989, Kifer et al., 1995] and *Gulog* [Dobbie and Topor,

1995] are more concerned with formalizing object-oriented concepts in a mathematical framework which could be used as a basis for building object-oriented databases or query systems. At this time, however, there is wide consensus that Kifer's F-logic is currently the most developed and complete formal model.

A major difference between object-oriented data models and *deductive* object-oriented data models has to do with how *method resolution* is accomplished [Abiteboul et al., 1993]. Whenever non-monotonic inheritance is permitted, there may be several implementations of the same method along a given path in the generalization hierarchy. Method resolution refers to the act of choosing one implementation of this method among the different implementations. In a procedural language, a method implementation (re)defined in a (sub)type defines a *total function* on its extension; i.e., this implementation of the method is applicable to every instance of the type. In this case, the choice of a particular method implementation is contingent only on the name of the method and the type of the instance to which the method is applied. Over-ridding is said to be *static*. In the case of deductive object-oriented data models, the rule definition of a method for a corresponding type is a *partial function*: a method defined on a type may or may not be applicable to an instance of that type, depending on whether or not the body of the method-rule evaluates to true. In this second scenario, over-ridding is said to be *dynamic*.

We clarify the distinction between static and dynamic over-ridding with the aid of an example. Suppose we have a generalization hierarchy where *working-student* is a subtype of *student*. The axiom:

$$X{:}student \vdash X[tax \rightarrow \mathtt{0}].$$

provides an implementation for the method *tax* and returns the value `0` when called on any instance X of *student*. Suppose "working students" are required to pay a flat-tax of `10` instead. This can be accomplished in our model by allowing the implementation of the method *tax* to be over-ridden in *working-student*:

$$X{:}working\text{-}student \vdash X[tax \rightarrow \mathtt{10}].$$

In this instance, over-ridding is said to be static since the new "implementation" of *tax* applies to all instances of *working-student*. On the other hand, if only students earning more than 2000 are required to pay taxes (proportional to the salaries they earn), we may write

$$X{:}\textit{working-student} \vdash X[\textit{tax} \rightarrow T] \leftarrow X[\textit{salary} \rightarrow S] \wedge S > 2000 \wedge T = 0.1 * S.$$

In this second implementation, over-ridding is dynamic since it constitutes a partial function on the set of all instances of *working-student*. For example, if we have

$$\textit{jack} : \textit{working-student}. \qquad \textit{jack}[\textit{salary} \rightarrow 3000].$$
$$\textit{jill} : \textit{working-student}. \qquad \textit{jill}[\textit{salary} \rightarrow 1500].$$

then *jack* will have to pay a tax of 300, whereas *jill* will pay no taxes (since the new implementation of the method *tax* does not apply to her).

4.2 Formalizing Context for Semantic Interoperability

In [McCarthy, 1987], McCarthy pointed out that statements about the world are never always true or false: the truth or falsity of a statement can only be understood with reference to a given *context*. This is formalized using assertions of the form:

$$\bar{c}: \qquad \textit{ist}(c, \sigma)$$

which suggests that the statement σ is true ("*ist*") in the context c[2], this statement itself being asserted in an *outer context* $\bar{c}$. *Lifting axioms*[3] are used to describe the

[2] In the words of Guha [Guha, 1991], contexts represents "the reification of the context dependencies of the sentences associated with the context." They are said to be "rich-objects" in that "they cannot be defined or completely described" [McCarthy and Hayes, 1987]. Consider, for instance, the context associated with the statement: "There are four billion people living on Earth". To fully qualify the sentence, we might add that it assumes that the time is 1991. However, this certainly is not the only relevant assumption in the underlying context, since there are implicit assumptions about who is considered a "live person" (are fetuses in the womb alive?), or what it means to be "on earth" (does it include people who are in orbit around the earth?)

[3] also called *articulation axioms* in Cyc/Carnot [Collet et al., 1991].

relationship between statements in different contexts. These statements are of the form

$$\bar{c}: \qquad ist(c, \sigma) \Leftrightarrow ist(c', \sigma')$$

which suggests that "σ in c states the same thing as σ' in c'".

McCarthy's notion of "contexts" and "lifting axioms" provide a useful framework for modeling statements in heterogeneous databases which are seemingly in conflict with one another. From this perspective, factual statements present in a data source are no longer "universal" facts about the world, but are true relative to the context associated with the source but not necessarily so in a different context. Thus, if we assign the labels c_1 and c_2 to contexts associated with sources 1 and 2 in Figure 3-1, we may now write:

$\bar{c}$: $ist(c_1,$ `r1('NTT', 1 000 000, 'JPY')`$)$.

$\bar{c}$: $ist(c_2,$ `r2('NTT', 5 000 000)`$)$.

The context $\bar{c}$ above refers to the ubiquitous context in which our discourse is conducted (i.e., the *integration context*) and may be omitted in the ensuing discussion whenever there is no ambiguity.

In the Context Interchange approach, the semantics of data are captured explicitly in a collection of statements asserted in the context associated with each source, while allowing conflict detection and reconciliation to be deferred to the time when a query is submitted. Building on the ideas developed in [Siegel and Madnick, 1991, Sciore et al., 1994], we would like to be able to represent the semantics of data at the level of individual data elements (as opposed to the predicate or sentential level), which allows us to identify and deal with conflicts at a finer level of granularity. Unfortunately, individual data elements may be present in a relation without a unique denotation. For instance, the value `1 000 000` in relation `r1` (as shown in Figure 3-1) simultaneously describes the revenue of `IBM` and `NTT` while being reported in different currencies and scale-factors. Thus, the statements

$ist(c_1,$ $currency(R, Y) \leftarrow$ `r1`$(N, R, Y))$.

$ist(c_1,\ scaleFactor(R, 1000) \leftarrow currency(R, Y),\ Y = \texttt{'JPY'}).$

$ist(c_1,\ scaleFactor(R, 1) \leftarrow currency(R, Y),\ Y \neq \texttt{'JPY'}).$

intending to represent the currencies and scale-factors of revenue amounts will result in multiple inconsistent values. To circumvent this problem, we introduce *semantic-objects*, which can be referenced unambiguously through their object-ids. Semantic-objects are complex terms constructed from the corresponding data values (also called *primitive-objects*) and are used as a basis for inferring about conflicts, but are never materialized in an object-store.

The data model underlying our integration approach, called COIN (for COntext INterchange), consists of both a *structural component* describing how data objects are organized, and a *language* which provides the basis for making formal assertions and inferences about a universe of discourse. In the remainder of this section, we provide a description of both of these components, followed by a formal characterization of the Context Interchange strategy in the form of a COIN framework. The latter will be illustrated with reference to the motivational example introduced in Chapter 3.

4.3 The Structural Elements of COIN

The COIN data model is a deductive object-oriented data model designed to provide explicit support for Context Interchange. Consistent with *object-orientation* [Atkinson et al., 1989], information units are modeled as *objects*, having unique and immutable *object-ids (oids)*, and corresponding to *types* in a *generalization hierarchy* with provision for *non-monotonic inheritance*. We distinguish between two kinds of data objects in COIN: *primitive objects*, which are instances of *primitive types*, and *semantic-objects* which are instances of *semantic-types*. Objects in COIN have both an oid and a value: these are identical in the case of primitive-objects, but different for semantic-objects. This is an important distinction which will become apparent shortly.

Primitive-types correspond to data types (e.g., strings, integers, and reals) which are native to sources and receivers. Semantic-types, on the other hand, are com-

plex types introduced to support the underlying integration strategy. Specifically, semantic-objects may have *properties* which are either attributes or modifiers. *Attributes* represent structural properties of the semantic-object under investigation: for instance, an object of the semantic-type *companyFinancials* must, by definition, describes some company; we capture this structural dependency by defining the attribute *company* for the semantic-type *companyFinancials*. *Modifiers*, on the other hand, are used as the basis for capturing "orthogonal" sources of variations concerning how the value of a semantic-object may be interpreted. Consider the semantic-type *moneyAmt*: the modifiers *currency* and *scaleFactor* defined for *moneyAmt* suggests two sources of variations in how the value corresponding to an instance of *moneyAmt* may be interpreted. "Orthogonality" here refers to the fact that the value which can be assigned to one modifier is independent of other modifiers, as is the case with *scaleFactor* and *currency*. This is not a limitation on the expressiveness of the model since two sources of variations which are correlated can always be modeled as a single modifier. As we shall see later, this simplification allows greater flexibility in dealing with conversions of values across different contexts.

Unlike primitive-objects, the *value* of a semantic-object may be different in different contexts. For example, if the (Skolem) term sk_0 is the oid for the object representing the revenue of `NTT`, it is perfectly legitimate for both

(1) $ist(c_1,\ value(sk_0,1\,000\,000))$; and

(2) $ist(c_2,\ value(sk_0,9\,600\,000))$,

to be true since contexts c_1 and c_2 embody different assumptions on what currencies and scale-factors are used to report the value of a revenue amount[4]. For our problem domain, it is often the case that the value of a semantic-object is known in some context, but not others. This is the case in the example above, where (1) is known, but not (2). The derivation of (2) is aided by a special lifting axiom defined below.

[4]A predicate-calculus language is used in the discussion here since it provides better intuition for most readers. The COIN language, for which properties are modeled as "methods" (allowing us to write $sk_0[value{\rightarrow}1\,000\,000]$ as opposed to $value(sk_0,1\,000\,000)$), will be formally defined in Section 4.4.

Definition 1 Let t be an oid-term corresponding to a semantic-object of the semantic-type τ, and suppose the value of t is given in context c_s. For any context represented by C, we have

$$ist(C,\ value(t, X) \leftarrow f_{\text{cvt}}(t, c_s, X') = X) \Leftrightarrow ist(c_s,\ value(t, X')).$$

We refer to f_{cvt} as the *conversion function* for τ in context C, and say that X is the value of t in context C, and that it is *derived from context* c_s. ❑

As we shall see later, the conversion function referenced above is polymorphically defined, being dependent on the type of the object to which it is applied, and may be different in distinct contexts.

Since modifiers of a semantic-type are orthogonal by definition, the conversion function referenced in the preceding definition can in fact be composed from other simpler conversion methods defined with reference to each modifier. To distinguish between the two, we refer to the first as a *composite conversion function*, and the latter as *atomic conversion functions*. Suppose modifiers of a semantic-type τ are $m_1, \ldots, m_k$, and f_{cvt} is a composite conversion function for τ. It follows that if t is an object of type τ, then

$$f_{\text{cvt}}(t, c_s, X') = X \text{ if } \exists X_1, \ldots, X_{k-1} \text{ such that}$$
$$(f^{(1)}_{\text{cvt}}(t, c_s, X') = X_1) \wedge \cdots \wedge (f^{(k)}_{\text{cvt}}(t, c_s, X_{k-1}) = X)$$

where $f^{(j)}_{\text{cvt}}$ corresponds to the atomic conversion function with respect to modifier m_j. Notice that the order in which the conversions are eventually effected need not correspond to the ordering of the atomic conversions imposed here, since the actual conversions are carried out in a *lazy* fashion and depends on the propagation of variable bindings.

Finally, we note that *value-based* comparisons in the relational model requires some adjustments here. We say that two semantic-objects are *distinct* if their oids are different. However, distinct semantic-objects may be *semantically-equivalent* as defined below.

Definition 2 Let $\oplus$ be a relational operator from the set $\{=, <, >, \leq, \geq, \neq, \ldots\}$. If t and t' are oid-terms corresponding to semantic-objects, then

$$(t \widetilde{\oplus} t') \Leftrightarrow (value(t, X) \wedge value(t', X') \wedge X \oplus X')$$

In particular, we say that t and t' are *semantically-equivalent* in context c if $ist(c, t \widetilde{=} t')$. ❑

We sometimes abuse the notation slightly by allowing primitive-objects to participate in semantic-comparisons. Recall that we do not distinguish between the oid and the value of a primitive object; thus, $ist(C,\ value(1\,000\,000,\ 1\,000\,000))$ is true irregardless of what C may be. Suppose we know that $ist(c_1,\ value(sk_0,\ 1\,000\,000))$, where sk_0 refers to the revenue of NTT as before. The expression

$$sk_0 \ \widetilde{<}\ 5\,000\,000$$

will therefore evaluate to "true" in context c_1 but not context c_2, since $ist(c_2,\ value(sk_0,\ 9\,600\,000))$. This latter fact can be derived from the value of sk_0 in c_1 (which is reported a priori in r_1) and the conversion function associated with the type *companyFinancials* (see Section 5.5).

4.4 The Language of COIN

We describe in this section the syntax and informal semantics of the language of COIN, which is inspired largely by *Gulog* [Dobbie and Topor, 1995][5]. Rather than making inferences using a *context logic* (see, for example, [Buvač, 1995]), we introduce "context" as first-class objects and capture variations in different contexts through the use of *parameterized methods*. For example, the context-formula $ist(c_1,\ value(sk_0,\ 1\,000\,000))$ can be equivalently written as $sk_0[value(c_1) \rightarrow 1\,000\,000]$ where $value(c_1)$ represents a (single-valued) method. This simplification is possible because of our

[5]Gulog differs from *F-logic* [Kifer et al., 1995] in that method rules are bound to the underlying types, which leads to different approaches for dealing with *non-monotonic inheritance*. Specifically, in the case of F-logic, it is not *rules* but *ground expressions* that are handed down the generalization hierarchy. Since we are interested in reasoning at the intensional level, the former model is more appropriate for us.

commitment to a common "vocabulary" (i.e., what types exists and what methods are applicable) and the fact that object ids remains immutable across different contexts. By writing statements which are fully *decontextualized* (i.e., "lifted" from the individual source and receiver contexts into the integration context), we are able to leverage on semantics and proof procedures developed without provision for contexts.

Following [Lloyd, 1987], we define an *alphabet* as consisting of (1) a set of *type symbols* which are partitioned into symbols representing semantic-types and primitive-types: each of which have a distinguished type symbol, denoted by $\top_{\mathcal{S}}$ and $\top_{\mathcal{P}}$ respectively; (2) an infinite set of *constant symbols* which represents the oids (or identically, values) of primitive-objects; (3) a set of *function symbols* and *predicate symbols*; (4) a set of *method symbols* corresponding to attributes, modifiers, and built-in methods (e.g., *value* and *cvt*); (5) an infinite set of *variables*; (6) the usual *logical connectives and quantifiers* $\wedge$, $\vee$, $\forall$, $\exists$, $\neg$, etc; (7) *auxiliary symbols* such as (,), [,], :, ::, $\rightarrow$, $\Rightarrow$ and so forth; and finally, (8) a set of *context symbols*, of the distinguished object-type called *ctx*, denoting contexts. A *term* is either a constant, a variable, or the token $f(t_1, \ldots, t_n)$ where f is a function symbol and $t_1, \ldots, t_n$ are terms. Since terms in our model refer to (logical) oids, they are called *oid-terms*. Finally, a predicate, function, or method symbol is said to be *n-ary* if it expects n arguments.

Definition 3 A *declaration* is defined as follows:

- if τ and τ' are type symbols, then $\tau :: \tau'$ is a *type declaration*. We say that τ is a *subtype* of τ', and conversely, that τ' is a *supertype* of τ. For any type symbol τ'' such that $\tau' :: \tau''$, τ is also a subtype of τ''.

- if t is a term and τ is a *type symbol*, then $t : \tau$ is an *object declaration*. We say that t is an *instance* of type τ. If τ' is a supertype of τ, then t is said to be of *inherited type* τ'.

- if p is an n-ary predicate symbol, and $\tau_1, \ldots, \tau_n$ are type symbols, then $p(\tau_1, \ldots, \tau_n)$ is a *predicate declaration*. We say that the *signature* of predicate p is $\tau_1 \times \cdots \times \tau_n$.

- if m is an attribute symbol and τ, τ' are symbols denoting semantic-types, then

$\tau[m \Rightarrow \tau']$ is an *attribute declaration*. We say that the signature of the attribute is $\tau \rightarrow \tau'$, and that the semantic-type τ has attribute m.

- if m is a modifier symbol, and τ, τ' are symbols denoting semantic-types, then $\tau[m(ctx) \Rightarrow \tau']$ is a *modifier declaration*. We say that m is a modifier of the semantic-type τ, which has signature $\tau \rightarrow \tau'$. Without any loss of generality, we assume that m is unique across all semantic-types.

- if τ is a semantic-type, and τ_1, τ_2 are primitive types, then $\tau[cvt(ctx)@ctx, \tau_1 \Rightarrow \tau_2]$ is a *compound conversion declaration*. We say that the signature of the compound conversion for τ is $\tau \times \tau_1 \rightarrow \tau_2$.

- if τ is a semantic-type, m is a modifier defined on τ, and τ_1, τ_2 are primitive types, then $\tau[cvt(ctx)@m, ctx, \tau_1 \Rightarrow \tau_2]$ is a *atomic conversion declaration*. We say that the signature of the atomic conversion of m for τ, is $\tau \times \tau_1 \rightarrow \tau_2$.

- if τ is a semantic-type, τ_1 is a primitive-type and c is a context symbol, then $\tau[value(ctx) \rightarrow \tau_1]$ is a *value declaration*. We say that the signature of the value for τ is given by $\tau \rightarrow \tau_1$.

Declarations for attributes, modifiers, conversions, and the built-in method *value* are collectively referred to as *method declarations*. ❑

Definition 4 An *atom* is defined as follows:

- if p is an n-ary predicate symbol with signature $\tau_1 \times \cdots \tau_n$ and $t_1, \ldots, t_n$ are of (inherited) type $\tau_1, \ldots, \tau_n$ respectively, then $p(t_1, \ldots, t_n)$ is a *predicate atom*.

- if m is an attribute symbol with signature $\tau \rightarrow \tau'$ and t, t' are of (inherited) types τ, τ' respectively, then $t[m \rightarrow t']$ is an *attribute atom*.

- if m is a modifier symbol with signature $\tau \rightarrow \tau'$, c is a context symbol, and t, t' are of (inherited) types τ, τ' respectively, then $t[m(c) \rightarrow t']$ is a *modifier atom*.

- if the compound conversion function for τ has signature $\tau \times \tau_1 \rightarrow \tau_2$, t, t_1, t_2 are of (inherited) types τ, τ_1, τ_2 respectively, c is a context symbol, and t_c is a context term, then $t[cvt(c)@t_c, t_1 \rightarrow t_2]$ is a *compound conversion atom.*

- if the atomic conversion atom of the modifier m has signature $\tau \times \tau_1 \rightarrow \tau_2$, c is a context symbol, t, t_1, t_2 are of (inherited) types τ, τ_1, τ_2 respectively, and t_c is a context term, then $t[cvt(c)@m, t_c, t_1 \rightarrow t_2]$ is a *atomic conversion atom* for m.

- if the value signature is given by $\tau \rightarrow \tau'$, c is a context symbol, and t, t' are of (inherited) types τ, τ_1, then $t[value(c) \rightarrow t']$ is a *value atom.*

As before, the atoms corresponding to attributes, modifiers, conversions, and built-in method *value* are referred to collectively as *method atoms.* ❑

Atoms can be combined to form *molecules* (or *compound atoms*): these are "syntactic sugar" which are notationally convenient, but by themselves do not increase the expressive power of the language. For example, we may write

- $t[m_1 \rightarrow t_1; \cdots; m_k \rightarrow t_k]$ as a shorthand for the conjunct $t[m_1 \rightarrow t_1] \wedge \cdots \wedge t[m_k \rightarrow t_k]$;

- $t[m \rightarrow t_1[m_1 \rightarrow t_2]]$ as a shorthand for $t[m \rightarrow t_1] \wedge t_1[m_1 \rightarrow t_2]$; and

- $t : \tau[m \rightarrow t']$ as a shorthand for $t : \tau \wedge t[m \rightarrow t']$.

Well formed formulas can be defined inductively in the same manner as in *first-order languages* [Lloyd, 1987]; specifically,

- an atom is a formula;

- if ϕ and φ are formulas, then $\neg\phi$, $\phi \wedge \varphi$ and $\phi \vee \varphi$ are all formulas;

- if ϕ is a formula and X is a variable occurring in ϕ, then both $(\forall X\, \phi)$ and $(\exists X\, \phi)$ are formulas.

Instead of dealing with the complexity of full-blown first-order logic, it is customary to restrict well-formed formulas to only *clauses.*

Definition 5 A *Horn clause* in the COIN language is a statement of the form

$, \vdash A \leftarrow B_1, \ldots, B_n$

where A can either be an atom or a declaration, and $B_1, \ldots, B_n$ is a conjunction of atoms. A is called the *head*, and $B_1, \ldots, B_n$ is called the *body* of the clause. If A is a method atom of the form $t[m@\ldots \rightarrow t']$ where t is a term denoting a semantic-object, then the *predeclaration* , must contain the object declarations for all oid-terms in the head. Otherwise, , may be omitted altogether. ❑

4.5 The COIN Framework

The COIN *framework* builds on the COIN data model to provide a formal characterization of the Context Interchange strategy for the integration of heterogeneous data sources.

Definition 6 A COIN *framework* $\mathcal{F}$ is a quintuple $<\mathcal{S}, \mu, \mathcal{E}, \mathcal{D}, \mathcal{C}>$ where

- $\mathcal{S}$, the *source set*, is a labeled multi-set $\{s_1 := S_1, \ldots, s_m := S_m\}$. The label s_i is the *name* of a source, and S_i consists of ground predicate atoms $r_{ij}(a_1, \ldots)$ as well as the *integrity constraints* which are known to hold on those predicates. The set of atoms of r_{ij} constitute a relation r_{ij} in s_i.

- μ, the *source-to-context mapping*, defines a (total) function from $\mathcal{S}$ to $\mathcal{C}$. If $\mu(s_i) = c_j$, we say that the source s_i is in context c_j.

- $\mathcal{D}$, the *domain model*, is a set consisting of declarations. Intuitively, declarations in the domain model identify the types, methods, and predicates which are known.

- $\mathcal{E}$, the *elevation set*, is a multi-set $\{E_1, \ldots, E_m\}$ where E_i is the set of *elevation axioms* corresponding to s_i in $\mathcal{S}$. E_i consists of three parts:

  - for each relation $r_{ij} \in S_i$, there is a clause which defines a corresponding *semantic-relation* r'_{ij} in which every primitive object in r_{ij} is replaced by a Skolem term in r'_{ij};

- for every oid-term in r'_{ij}, we identify its type via the introduction of an object declaration, and define the values which are assigned to structural properties (i.e., attributes); and
- for every oid-term in r'_{ij}, we define its value in context $c(= \mu(s_i))$ with reference to r_{ij}.

- $\mathcal{C}$, the *context multi-set*, is a labeled multi-set $\{c_1 := C_1, \ldots, c_n := C_n\}$ where c_i is a context symbol, and C_i, called the *context set* for c_i, is set of clauses which provides a description of the relevant data semantics in context c_i. ❑

We provide the intuition for the above definition by demonstrating how the integration scenario shown in Figure 3-1 can be represented in a COIN framework $\mathcal{F} = <\mathcal{S}, \mu, \mathcal{E}, \mathcal{D}, \mathcal{C}>$. Figures 4-1 and 4-2 present a partial codification which we will elaborate briefly below:

- The contents of the source set $\mathcal{S}$ is simply the set of ground atoms present in the data sources. We place no limitation on the number of relations which may be present in each source; in the current example, it happens that each source has only one relation. The rules following the ground atoms are functional dependencies which are known to be true in the respective relation. For instance, the two rules in s_1 defines the functional dependency `cname` $\rightarrow$ {`revenue`,`currency`} on the attributes in r_1.

- The function μ is defined as a relation on $\mathcal{S} \times \mathcal{C}$: thus, source s_1 is mapped to context c_1, whereas s_2 and s_3 are both mapped to context c_2.

- The domain model $\mathcal{D}$ consists of two parts. The left-half (as seen in Figure 4-1) identifies (1) the semantic-types which are known and the generalization hierarchy; (2) the declarations for methods which are applicable to the semantic-types; and the signatures for the predicates corresponding to the semantic relations (r'_i). The right-half does the same for primitive-types and predicates for the extensional relations.

Source set $\mathcal{S}$

$s_1 := \{\ r_1(\texttt{'IBM'}, 1\,000\,000, \texttt{'USD'}).\ r_1(\texttt{'NTT'}, 1\,000\,000, \texttt{'JPY'}).$
$R_1 = R_2 \leftarrow r_1(N, R_1, \_), r_1(N, R_2, \_).$
$Y_1 = Y_2 \leftarrow r_1(N, \_, Y_1), r_1(N, \_, Y_2).\quad \}$

$s_2 := \{\ r_2(\texttt{'IBM'}, 1\,500\,000).\ r_2(\texttt{'NTT'}, 5\,000\,000).$
$E_1 = E_2 \leftarrow r_2(N, E_1), r_2(N, E_2).\quad \}$

$s_3 := \{\ r_3(\texttt{'USD'}, \texttt{'JPY'}, 104.0).\ r_3(\texttt{'JPY'}, \texttt{'USD'}, 0.0096).$
$T_1 = T_2 \leftarrow r_3(X, Y, T_1), r_3(X, Y, T_2).\quad \}$

Source-to-Context Mapping μ

$\{\mu(s_1, c_1), \mu(s_2, c_2), \mu(s_3, c_2)\}$

Domain model $\mathcal{D}$

/ type declarations */*

semanticNumber :: $\top_S$. — *number* :: $\top_P$.
semanticString :: $\top_S$. — *varchar* :: $\top_P$.
moneyAmt :: *semanticNumber*. — *integer* :: *number*.
companyFinancials :: *moneyAmt*. — *real* :: *number*.
currencyType :: *semanticString*.
companyName :: *semanticString*.

/ attribute declaration */*

moneyAmt[*company* ⇒ *companyName*].

/ modifier declarations */*

moneyAmt[*currency(ctx)*⇒ *currencyType*; *scaleFactor(ctx)*⇒ *semanticNumber*].

/ value declarations */*

semanticString[*value(ctx)*⇒ *varchar*].
semanticNumber[*value(ctx)*⇒ *number*].

/ conversion declarations */*

semanticString[*cvt(ctx)@ctx,varchar* ⇒ *varchar*].
semanticNumber[*cvt(ctx)@ctx,number* ⇒ *number*].
moneyAmt[*cvt(ctx)@ctx,number* ⇒ *number*].
moneyAmt[*cvt(ctx)@ctx,scaleFactor,number* ⇒ *number*].
moneyAmt[*cvt(ctx)@ctx,currency,number* ⇒ *number*].

/ predicate declarations */*

r_1'(*companyName,companyFinancials,currencyType*). — r_1(*varchar,integer,varchar*).
r_2'(*companyName,companyFinancials*). — r_2(*varchar,integer*).
r_3'(*currencyType,currencyType,semanticNumber*). — r_3(*varchar,varchar,real*).

Figure 4-1: The source set, source-to-context mapping, and domain model for the COIN framework corresponding to the motivational example.

- The first clause in each E_i of the elevation set $\mathcal{E}$ defines the semantic relation r_i' corresponding to the relation r_i; the semantic relations are defined on semantic-objects (as opposed to primitive-objects), which are instantiated as Skolem terms. The Skolem function (e.g., $f_{\text{r2\#expenses}}$) are chosen in the way such that when applied to the key-value of a tuple in the corresponding relation (e.g., `'NTT'`), the resulting Skolem term (i.e., $f_{\text{r2\#expenses}}$(`'NTT'`)) would in fact identify a unique "cell" in the relation as shown in Figure 3-2 (in this case, the expenses of `NTT` as reported in relation r_2).

- Object declarations and attribute atoms in the elevation set provide a way of specifying the types of corresponding Skolem terms introduced in the semantic relation. For instance, any Skolem term $f_{\text{r1\#revenue}}(\_)$ is asserted to be an instance of the semantic-type *companyFinancials*. The attribute atom following this declaration defines the object that is assigned to the *company* attribute for this semantic-object

- The values of the Skolem terms introduced in the semantic relation are defined through the clauses shown last. The primitive-objects assigned are obtained directly from the extensional relation. Clearly, the value assignment is valid only within the context of the source as identified by μ; the values of the Skolem terms in a different context can be derived through the use of conversion functions, which we will define later.

The context multi-set $\mathcal{C}$ is given by $\{c_1 := C_1, c_2 := C_2\}$ and is defined by the axioms shown in Figure 4-3. There are two kinds of axioms: modifier value definitions and conversion definitions.

Consistent with our data model, modifiers can be assigned different values in distinct contexts: this constitutes the principle mechanism for describing the meaning of data in disparate contexts. For example, the fact that in context c_1, *companyFinancials* are reported using a scale-factor of `1000` whenever it is reported in `JPY`, and `1` otherwise, can be represented by the formula:

$\forall X' : \textit{companyFinancials}\ \exists F' : \textit{number}\ \vdash$

Elevation Axioms E_1 of $\mathcal{E}$

$r_1'(f_{\text{r1\#cname}}(X_1), f_{\text{r1\#revenue}}(X_1), f_{\text{r1\#currency}}(X_1)) \leftarrow r_1(X_1, \_, \_).$
$f_{\text{r1\#cname}}(\_) : companyName.$
$f_{\text{r1\#revenue}}(\_) : companyFinancials.$
$f_{\text{r1\#revenue}}(X_1)[company \rightarrow f_{\text{r1\#cname}}(X_1)].$
$f_{\text{r1\#currency}}(\_) : currencyType.$
$f_{\text{r1\#cname}}(X_1)[value(C) \rightarrow X_1] \leftarrow r_1(X_1, \_, \_), \mu(s_1, C).$
$f_{\text{r1\#revenue}}(X_1)[value(C) \rightarrow X_2] \leftarrow r_1(X_1, X_2, \_), \mu(s_1, C).$
$f_{\text{r1\#currency}}(X_1)[value(C) \rightarrow X_3] \leftarrow r_1(X_1, \_, X_3), \mu(s_1, C).$

Elevation Axioms E_2 of $\mathcal{E}$

$r_2'(f_{\text{r2\#cname}}(X_1), f_{\text{r2\#expenses}}(X_1)) \leftarrow r_2(X_1, \_).$
$f_{\text{r2\#cname}}(\_) : companyName.$
$f_{\text{r2\#expenses}}(\_) : companyFinancials.$
$f_{\text{r2\#expenses}}(X_1)[company \rightarrow f_{\text{r2\#cname}}(X_1)].$
$f_{\text{r2\#cname}}(X_1)[value(C) \rightarrow X_1] \leftarrow r_2(X_1, \_), \mu(s_2, C).$
$f_{\text{r2\#expenses}}(X_1)[value(C) \rightarrow X_2] \leftarrow r_2(X_1, X_2), \mu(s_2, C).$

Elevation Axioms E_3 of $\mathcal{E}$

$r_3'(f_{\text{r3\#fromCur}}(X_1, X_2), f_{\text{r3\#toCur}}(X_1, X_2), f_{\text{r3\#exchangeRate}}(X_1, X_2)) \leftarrow r_3(X_1, X_2, \_).$
$f_{\text{r3\#fromCur}}(\_, \_) : currencyType.$
$f_{\text{r3\#toCur}}(\_, \_) : currencyType.$
$f_{\text{r3\#exchangeRate}}(\_, \_) : semanticNumber.$
$f_{\text{r3\#fromCur}}(X_1, X_2)[value(C) \rightarrow X_1] \leftarrow r_3(X_1, X_2, \_), \mu(s_3, C).$
$f_{\text{r3\#toCur}}(X_1, X_2)[value(C) \rightarrow X_2] \leftarrow r_3(X_1, X_2, \_), \mu(s_3, C).$
$f_{\text{r3\#exchangeRate}}(X_1, X_2)[value(C) \rightarrow X_3] \leftarrow r_3(X_1, X_2, X_3), \mu(s_3, C).$

Figure 4-2: Elevation set corresponding to the motivational example

Context c_1:

/* *modifier value assignments* */

X' : *companyFinancials* $\vdash$ $X'[scaleFactor(c_1) \rightarrow scaleFactor(c_1, X')]$.

X' : *companyFinancials*, $scaleFactor(c_1, X')$: *number* $\vdash$

$scaleFactor(c_1, X')[value(c_1) \rightarrow \texttt{1}] \leftarrow X'[currency(c_1) \rightarrow Y'], Y' \overset{c_1}{\not\cong} \texttt{'JPY'}$.

X' : *companyFinancials*, $scaleFactor(c_1, X')$: *number* $\vdash$

$scaleFactor(c_1, X')[value(c_1) \rightarrow \texttt{1 000}] \leftarrow X'[currency(c_1) \rightarrow Y'], Y' \overset{c_1}{\cong} \texttt{'JPY'}$.

X' : *companyFinancials* $\vdash$ $X'[currency(c_1) \rightarrow currency(c_1, X')]$.

X' : *companyFinancials*, $currency(c_1, X')$: *currencyType* $\vdash$

$currency(c_1, X')[value(c_1) \rightarrow Y] \leftarrow X'[company \rightarrow N_0'], r_1'(N_1', R', Y'), N_0' \overset{c_1}{\cong} N_1',$
$Y'[value(c_1) \rightarrow Y]$.

/* *conversion function definitions* */

X' : *moneyAmt* $\vdash$

$X'[cvt(c_1)@C, U \rightarrow V] \leftarrow X'[cvt(c_1)@scaleFactor, C, U \rightarrow W],$
$X'[cvt(c_1)@currency, C, W \rightarrow V]$.

X' : *moneyAmt* $\vdash$

$X'[cvt(c_1)@scaleFactor, C, U \rightarrow V] \leftarrow X'[scaleFactor(c_1) \rightarrow \_[value(c_1) \rightarrow F_1]],$
$X'[scaleFactor(C) \rightarrow \_[value(c_1) \rightarrow F]], V = U * F/F_1$

X' : *moneyAmt* $\vdash$

$X'[cvt(c_1)@currency, C, U \rightarrow V] \leftarrow X'[currency(c_1) \rightarrow Y_1'], X'[currency(C) \rightarrow Y'],$
$Y_1' \overset{c_1}{\cong} Y', V = U.$

X' : *moneyAmt* $\vdash$

$X'[cvt(c_1)@currency, C, U \rightarrow V] \leftarrow X'[currency(c_1) \rightarrow Y_1'], X'[currency(C) \rightarrow Y'],$
$Y_1' \overset{c_1}{\not\cong} Y', r_3'(Y_f', Y_t', R') , Y_f' \overset{c_1}{\cong} Y_1', Y_t' \overset{c_1}{\cong} Y',$
$R'[value(c_1) \rightarrow R], V = U * R.$

Context c_2:

/* *modifier value assignments* */

X' : *companyFinancials* $\vdash$ $X'[scaleFactor(c_2) \rightarrow scaleFactor(c_2, X')]$.

X' : *moneyAmt*, $scaleFactor(c_2, X')$: *number* $\vdash$ $scaleFactor(c_2, X')[value(c_2) \rightarrow \texttt{1}]$.

X' : *companyFinancials* $\vdash$ $X'[currency(c_2) \rightarrow currency(c_2, X')]$.

X' : *moneyAmt*, $currency(c_2, X')$: *currencyType* $\vdash$

$currency(c_2, X')[value(c_2) \rightarrow \texttt{'USD'}]$.

/* *conversion definitions are similar to* c_1 *and omitted for brevity* */

Figure 4-3: Context sets for $\mathcal{C}$ for the motivational example at hand.

$$
\begin{aligned}
&(\, X'[\mathit{scaleFactor}(c_1){\rightarrow}F']\,)\wedge \\
&(\, F'[\mathit{value}(c_1){\rightarrow}\ \mathtt{1\,000}] \leftarrow X'[\mathit{currency}(c_1){\rightarrow}Y'] \wedge Y' \overset{c_1}{\cong} \mathtt{'JPY'}\,) \wedge \\
&(\, F'[\mathit{value}(c_1){\rightarrow}\ \mathtt{1}] \leftarrow X'[\mathit{currency}(c_1){\rightarrow}Y'] \wedge Y' \overset{c_1}{\not\cong} \mathtt{'JPY'}\,).
\end{aligned}
$$

The above formula is not in clausal form, but can be transformed to definite Horn clauses by Skolemizing the existentially quantified variable F'. For example, the above formulas can be reduced to the following clauses:

$$
\begin{aligned}
&X' : \mathit{companyFinancials} \vdash \\
&\qquad X'[\mathit{scaleFactor}(c_1){\rightarrow}f_{\mathit{scaleFactor}(c_1)}(X')]. \\
&X' : \mathit{companyFinancials},\ f_{\mathit{scaleFactor}(c_1)}(X') : \mathit{number} \vdash \\
&\qquad f_{\mathit{scaleFactor}(c_1)}(X')[\mathit{value}(c_1){\rightarrow}\ \mathtt{1\,000}] \leftarrow X'[\mathit{currency}(c_1){\rightarrow}Y'], Y' \overset{c_1}{\cong} \mathtt{'JPY'}. \\
&X' : \mathit{companyFinancials},\ f_{\mathit{scaleFactor}(c_1)}(X') : \mathit{number} \vdash \\
&\qquad f_{\mathit{scaleFactor}(c_1)}(X')[\mathit{value}(c_1){\rightarrow}\ \mathtt{1}] \leftarrow X'[\mathit{currency}(c_1){\rightarrow}Y'], Y' \overset{c_1}{\not\cong} \mathtt{'JPY'}.
\end{aligned}
$$

where $f_{\mathit{scaleFactor}(c_1)}$ is a unique Skolem function; for notational simplicity, we replace $f_{\mathit{scaleFactor}(c_2)}(X')$ with the term $\mathit{scaleFactor}(c_2, X')$. Currency values corresponding to instances of *companyFinancials* are obtained directly from the extensional relation r_1 as shown in Figure 4-3. In this instance, it is necessary to reference an extensional relation because "meta-data" are represented along with "data" in a source. In a "better-behaved" situation (such as context c_2), the modifier values for *currency* and *scaleFactor* can be defined independently of the underlying schema. It is worthwhile to note that our framework is sufficiently expressive to capture both types of scenario, although the first tends to make the boundary between intensional and extensional knowledge more fuzzy.

Conversion functions define how the value of a given semantic-object can be derived in the current context, given that its value is known with respect to a different context. As shown in Figure 4-3, the first clause in the group (for context c_1) defines the conversion for *moneyAmt* via the composition of atomic conversion functions for *scaleFactor* and *currency*. The *scaleFactor* conversion is defined by identifying the respective scale-factors in the source and target contexts and multiplying the value of the *moneyAmt* object by the ratio of the two. The *currency* conversion is obtained by

multiplying the source value by a conversion rate which is obtained via a lookup on yet another data source (r_3). Notice that these conversions are defined with respect to *moneyAmt* but are applicable to *companyFinancials* via behavioral inheritance of the methods. In general, the repertoire of conversion functions can be extended arbitrarily by defining the conversion externally and invoking the external functions using the built-in **system** predicate which serves as an escape hatch to the operating system. However, encapsulating the conversion in external functions makes it harder to reason about the properties of the conversion; for example, the explicit treatment of arithmetic operators and table-lookups (in conversion functions) allow us to exploit opportunities for optimization, say, by rewriting the arithmetic expression to reduce the size of intermediary tables during query execution[6].

4.6 A Meta-Logical Extension to the COIN Framework

In Section 4.5, context knowledge in a COIN framework is represented by a set of separate theories (i.e., $\mathcal{C} = \{c_1 := C_1, \ldots, c_n := C_n\}$). We describe here an extension to this basic framework which allows new contexts to be defined in terms of existing ones in an incremental fashion. Two basic mechanisms underly this move to such an extension: the treatment of context as a set of parameterized statements and the introduction of the hierarchical operator $\prec$, which defines a *subcontext* relation on the set $\{c_1, \ldots, c_n\}$.

Recall that the relative truth or falsity of a statement can be represented using McCarthy's *ist*, so that

$$ist(c_i, \sigma)$$

is taken to mean that the statement σ is true in context c_i. The relation $\prec$ allows us to make incremental refinements to statements which describe what is already known

[6]Details of query optimization strategies that take into account conversion functions are beyond the scope of the work reported here. A more detailed discussion can be found in [Daruwala et al., 1995].

about an enclosing context. Thus, if c_i is a *subcontext* of c_j, denoted by $c_i \prec c_j$, this allows us to introduce a *differential context* denoted by δ_{c_i}, such that:

$$ist(c_i, \sigma) \leftarrow \sigma \in \delta_{c_i}$$
$$ist(c_i, \sigma) \leftarrow c_i \prec c_j,\ ist(c_j, \sigma),\ \textit{not-overridden}(\delta_{c_i}, \sigma).$$

The predicate *not-overridden* indicates that the statement σ obtained from the more general context c_j is not explicitly overridden by the differential context. The composition of a new context theory of c_i from c_j and δ_{c_i} is similar to that accomplished by the `isa` operator defined in [Borgi et al., 1990].

In the COIN data model, statements in a context are "decontextualized" by making explicit references to its reification in the form of a context-object. For example, the statement

$$ist(c_j, t[m_1 \rightarrow t'] \leftarrow t[m_2 \rightarrow t']).$$

can be equivalently stated as

$$t[m_1(c_j) \rightarrow t'] \leftarrow t[m_2(c_j) \rightarrow t'].$$

This second form simplifies the inferences which are undertaken to support context mediation, but requires some adjustment to allow statements to be inherited. Specifically, if the above statement is inherited by context c_i ($\prec c_j$), we will need to replace the references to c_j with c_i. This is accomplished by requiring all statements in δ_{c_j} to be parameterized; i.e.,

$$\delta_{c_j}(X) = \{\sigma_1(X), \ldots, \sigma_l(X)\}$$

For instance, the earlier statement would have been asserted as

$$\sigma(X) = t[m_1(X) \rightarrow t'] \leftarrow t[m_2(X) \rightarrow t'].$$

in the set δ_{c_j}. The statement $\sigma(X)$ is said to be *uninstantiated*. The collection of uninstantiated axioms forms an *uninstantiated context set*.

Definition 7 Let $\delta\mathcal{C} = \{c_0 := C_0(X), \delta_{c_1}(X), \ldots, \delta_{c_n}(X)\}$, for which $\delta_{c_i}(X)$ $(i = 1, \ldots, n)$ is said to be the *differential* for context c_i with respect to $\prec$, which defines a partial order on the contexts $\{c_1, \ldots, c_n\}$. Let $\{c_{i_1}, \ldots, c_{i_k}\}$ be the predecessors of c_i with respect to the subcontext relation $\prec$. Then the uninstantiated context set for c_{i_j}, denoted by $C_{i_j}(X)$, can be obtained from $C_i(X)$ as before: i.e.,

- $\sigma(X) \in C_{i_j}(X) \leftarrow \sigma(X) \in \delta_{c_{i_j}}(X)$
- $\sigma(X) \in C_{i_j}(X) \leftarrow \sigma(X) \in C_i(X)$, *not-overridden*$(\delta_{c_{i_j}}(X), \sigma(X))$.

The context c_0 is said to be the *default context* and forms the basis for the other differentials. ❑

Definition 8 Given $\delta\mathcal{C} = \{c_0 := C_0(X), \delta_{c_1}(X), \ldots, \delta_{c_n}(X)\}$ and the subcontext relation $\prec$. Suppose $C_i(X)$ is the uninstantiated context set for c_i obtained inductively using Definition 7. The context set for c_i is given by the set $C_i(c_i)$. ❑

Notice that we have not described how one is to determine whether or not a given statement is being overridden in a specific context. The simplest approach is to assume that whenever a method atom appears in the head in a differential context set, none of the other rules (pertaining to this method) defined in any of its supercontext applies. For example, if the *scaleFactor* for the type *companyFinancials* is given in two distinct context differentials along a given path in the hierarchy, then the statement in the more specific context is said to take precedence and will be used in the corresponding context.

The above scheme leads to the following extended formulation of a COIN framework.

Definition 9 The *extended* COIN framework is a sextuple given by $<\mathcal{S}, \mu, \mathcal{E}, \mathcal{D}, \delta\mathcal{C}, \prec >$, where $\mathcal{S}, \mathcal{E}$, and $\mathcal{D}$ are defined as before in Definition 6, $\delta\mathcal{C}$ is as defined in Definition 7, and $\prec$ is the subcontext relation defined on the set of contexts $\{c_1, \ldots, c_n\}$ induced by $\delta\mathcal{C}$. ❑

Chapter 5

Query Answering in the COIN Framework

Following the same algorithm outlined in [Abiteboul et al., 1993], any collection of COIN clauses can be translated to *Datalog with negation (Datalog^{neg})* (or equivalently, *normal Horn program* [Lloyd, 1987]), for which the semantics as well as computation procedures have been widely studied [Ullman, 1991b][1]. In this Chapter, we explore an alternative approach based on *abductive* reasoning. The *abductive framework* provides us with *intensional* (as opposed to *extensional*) answers to a query[2]. We describe this abductive framework below and the relationship between query mediation in a COIN framework and query answering in an abductive framework. We assume some familiarity with logic programming at the level of [Lloyd, 1987] in the ensuing discussion, and for most part, shall remain faithful to the notations therein.

[1] The fact that "object-based logics" can be encoded in classical predicate logic has been known for a long time (see for example, [Chen and Warren, 1989]). This however should not cause us to "lose faith" in our data model, since the syntax of the language plays a pivotal role in shaping our conceptualization of the problem and in finding solutions at the appropriate levels of abstraction.

[2] This change in perspective is beneficial for a variety of reasons (see Chapter 3), and will not be repeated here.

5.1 Abductive Logic Programming

Abduction [Kowalski, 1990] refers to a particular kind of hypothetical reasoning which, in the simplest case, takes the form:

> From observing A and the axiom $B \rightarrow A$
>
> Infer B as a possible "explanation" of A.

For instance, given the axioms (1) "it rained" $\rightarrow$ "the floor is wet", and (2) "sprinkler was on" $\rightarrow$ "the floor is wet", the observation of a "wet floor" will lead us to conclude that "it rained" or that "sprinkler was on". As illustrated in this example, abduction is typically used for identifying explanations for observations. It is a form of non-monotonic reasoning, because explanations which are consistent with one state of a knowledge base (theory) may be inconsistent with new information. Hence, if we had known that it did not rain, then the explanation "it rained" will have to be retracted. Abductive reasoning has been used in diagnostic reasoning (e.g., as in medical diagnosis where observations are symptons to be explained and explanations sought are the diseases a patient may be infected with), high level vision, natural language understanding, planning, knowledge assimilation, and database view updates [Kakas et al., 1993, and references therein].

Following Eshghi and Kowalski [1989], we define an *abductive framework* to be a triple $<\mathcal{T}, \mathcal{A}, \mathcal{I}>$ where $\mathcal{T}$ is a theory, $\mathcal{I}$ is a set of integrity constraints, and $\mathcal{A}$ is a set of predicate symbols, called *abducible* predicates. Given an abductive framework $<\mathcal{T}, \mathcal{A}, \mathcal{I}>$ and a sentence $\exists \vec{X} q(\vec{X})$ (the *observation*), the *abductive task* can be characterized as the problem of finding a *substitution* θ and a set of abducibles Δ, called the *abductive explanation* for the given observation, such that

(1) $\mathcal{T} \cup \Delta \models q(\vec{X})\theta$,

(2) $\mathcal{T} \cup \Delta$ satisfies $\mathcal{I}$; and

(3) Δ has some properties that make it "interesting".

This characterization of abduction is independent of the language in which the sentences are formulated. Requirement (1) states that Δ, together with $\mathcal{T}$, must be

capable of providing an explanation for the observation $q(\vec{X})\theta$. The consistency requirement in (2) distinguishes abductive explanations from inductive generalizations. Finally, in the characterization of Δ in (3), "interesting" means primarily that literals in Δ are atoms formed from abducible predicates: where there is no ambiguity, we refer to these atoms also as *abducibles*. In most instances, we would like Δ to also be minimal or non-redundant.

Abductive logic programming (ALP) is the extension of *logic programming* [Lloyd, 1987] to support abductive reasoning. *Semantics* and *proof procedures* for ALP have been proposed by Kakas and Mancarella [1990b], Console et al. [1991], Toni [1995], and Denecker and Schreye [1992a,b]. As pointed out in [Kakas et al., 1993], the relationship between semantics and proof procedures can be understood as a relationship between program specifications and programs. A program specification characterises what is the intended result expected from the execution of the program. In the same way, the semantics of an abductive logic program can be understood as an abstract definition of what is to be computed by a proof procedure; from this perspective, semantics is not so much concerned with explicating meaning in terms of truth and falsity, as it is with providing an abstract specification which "declaratively" expresses what we want to compute. The various proof procedures proposed in the literature are appropriate for different semantics and considers logic programs with different expressivity. For example, the abductive procedures described in [Cox and Pietrzykowski, 1986] and [Finger and Genesereth, 1985] work only for *definite* logic programs (i.e., without negation), whereas the abduction procedure defined in [Kakas and Mancarella, 1990a] allows a literal to be abducted only if it is ground.

It has been shown that in general, the abductive task is NP-hard [Eiter and Gottlob, 1993] even if we restrict the theory to propositional clauses. Eshghi [1993], however, has shown that if the theory comprises of Horn propositional clauses with additional restrictions, then there exists a polynomial time algorithm for finding minimal explanations. The generalization of these results to predicate logic is not yet clear and warrants further study.

5.2 On the Relationship between Abduction and Deduction

In [Kowalski, 1991], Kowalski pointed out that abductive solutions to a query can also be obtained by *deduction*. Consider, for instance, the theory $\mathcal{T}$ defined as follows:

$$\begin{aligned}\mathcal{T} = \{\ & \textit{wobbly-wheel} \leftarrow \textit{flat-tire}.\\ & \textit{wobbly-wheel} \leftarrow \textit{broken-spokes}.\\ & \textit{flat-tire} \leftarrow \textit{punctured-tube}.\\ & \textit{flat-tire} \leftarrow \textit{leaky-valve}.\quad \}\end{aligned}$$

where $\mathcal{A} = \{\textit{broken-spokes, punctured-tubes, leaky-valve}\}$. Given the abductive query $Q = \leftarrow \textit{wobbly-wheel}$, we will obtain the following three abductive answers:

$$\Delta_1 = \{\ \textit{punctured-tube}\ \}$$
$$\Delta_2 = \{\ \textit{leaky-valve}\ \}$$
$$\Delta_3 = \{\ \textit{broken-spokes}\ \}$$

The same answers, however, can be obtained using deduction by considering the theory $\mathcal{T}'$ obtained by taking the only-if part of every definition of a non-abducible predicate in the Clark-completion [Clark, 1978] of $\mathcal{T}$ and by adding the negation of Q. For the example at hand, this yields the theory $\mathcal{T}'$ given by

$$\begin{aligned}\mathcal{T}' = \{\ & \textit{flat-tire} \vee \textit{broken-spokes} \leftarrow \textit{wobbly-wheel}.\\ & \textit{punctured-tube} \vee \textit{leaky-valve} \leftarrow \textit{flat-tire}.\\ & \textit{wobbly-wheel} \leftarrow.\quad \}\end{aligned}$$

It is easy to see that the program $\mathcal{T}'$ has only three minimal models given by:

$$M_1 = \{\ \textit{wobbly-wheel, flat-tire, punctured-tube}\ \}$$
$$M_2 = \{\ \textit{wobbly-wheel, flat-tire, leaky-valve}\ \}$$
$$M_1 = \{\ \textit{wobbly-wheel, broken-spokes}\ \}$$

These corresponds exactly to the abductive solutions obtained earlier if they are restricted to only abducible predicates. As was pointed out in Console et al. [1991],

and subsequently Denecker and Schreye [1992a], the duality between abduction and deduction holds in the general case and not just only when the theory is propositional (i.e., variable-free). Denecker [1993] went further to point out that when a theory is "complete" (in the sense that every formula or its negation is a logical consequence of the theory), abduction then collapses to deduction.

5.3 The SLD+Abduction Procedure and Its Extensions

We describe in this section an abduction procedure based on extension to SLD resolution, called *SLD+Abduction*. The underlying idea is first reported in [Cox and Pietrzykowski, 1986], and has inspired various different extensions. The account we give here follows that in [Shanahan, 1989].

We first consider *SLD resolution*[3]. Given a theory $\mathcal{T}$ consisting of Horn clauses and a goal clause $\leftarrow G$, and *SLD-refutation* of $\leftarrow G$ is a sequence of goal clauses $\leftarrow G_0(=G); \leftarrow G_1; \cdots; \leftarrow G_n$ where $\leftarrow G_n$ is the empty clause and each $\leftarrow G_{i+1}$ is obtained from $\leftarrow G_i$ by resolving one of its literals (the *selected literal*) with one of the clauses in $\mathcal{T}$. The Prolog interpreter, for instance, implements a form of SLD resolution where the leftmost literal in a goal clause is always selected. Since there may be many clauses in $\mathcal{T}$ which can be resolved with the selected literal, a space of possible refutations is defined (in the form of an *SLD-tree*). The search space defined by an SLD-tree may be searched in a number of ways. For example, this is accomplished in Prolog in a depth-first manner, resulting in chronological backtracking.

Suppose now that there is some $\leftarrow G_i$, whose selected literal g will not resolve with any clause in $\mathcal{T}$. This means that the part of the subtree with $\leftarrow G_i$ at the root is not worth exploring any further, since it will not contain any branch that leads

[3]SLD resolution stands for SL resolution for Definite clauses. SL stands for Linear resolution with Selection function. The term *LUSH-resolution* is sometimes used to refer to the same thing. In the description which follows, we assume some familiarity with logic programming at the level of [Lloyd, 1987, Chapters 1 and 2].

to a refutation (i.e., one which terminates in an empty clause). Given however that we are searching for a set of unit clauses Δ, such that $\mathcal{T} \cup \Delta \models G$, then clearly by letting Δ include a unit clause which resolves with g, we can continue the search with $\leftarrow G_{i+1}$, which is obtained from $\leftarrow G_i$ minus the literal g. This observation forms the basis for the SLD+Abduction procedure which we proceed to describe below.

Given an abductive framework $<\mathcal{T}, \mathcal{A}, \mathcal{I}>$ and the abductive query $q(\vec{X})$, consider the sequence given by

$\leftarrow G_0, \Delta_0$ where $G_0 = q(\vec{X})$ and Δ_0 is the empty set

$\vdots$

$\leftarrow G_n, \Delta_n$

such that G_{i+1}, Δ_{i+1} is derived from G_i, Δ_i as follows:

- if g, the selected literal of $\leftarrow G_i$, can be resolved with a clause in $\mathcal{T}$, then a single resolution step is taken to yield G_{i+1}, and $\Delta_{i+1} = \Delta_i$;
- if g is abducible, g' is g with all its variables replaced by Skolem constants, and $\mathcal{T} \cup \Delta_i \cup \{g' \leftarrow\}$ is consistent with $\mathcal{I}$, then G_{i+1} is G_i less g, and $\Delta_{i+1} = \Delta_i \cup \{g' \leftarrow\}$.

The sequence obtained is said to be a *derivation* of G with respect to the abductive framework $<\mathcal{T}, \mathcal{A}, \mathcal{I}>$. A derivation, as we have just defined, is said to be a *refutation* if $\leftarrow G_n$ is the empty clause. The accumulated set of unit clauses Δ_n is said to be the *residue* corresponding to this refutation, and constitutes the abductive answer to $\forall\,(\,q(\vec{X})\theta\,)$, where θ is the substitution obtained from the composition of all substitutions leading to the refutation, restricted to the variables $\vec{X}$.

In the abduction step above, we require that the selected literal g to be Skolemized. This is because variables in the unit clause "$g' \leftarrow$" needs only be existentially quantified for it to be resolvable with g. If the Skolemization is not done, the abducted fact "$g' \leftarrow$" (where $g' = g$) would have been unnecessarily strong. This Skolemization, however, introduces additional complexity since it becomes necessary to deal with equality constraints on Skolem constants. This is due to the fact that a Skolem

constant (sk) introduced earlier in the SLD-derivation (say in $\leftarrow G_i$) may have to be unified with a specific term (t) later on (in $\leftarrow G_j$, where $j > i$). Eshghi [1988] suggested that this can be dealt with by introducing the equality predicate as an abducible predicate and to add the theory of *Free Equality (FEQ)* [Clark, 1978] as integrity constraints. Thus, when a Skolem constant sk is to be unified with a term t, the equality fact $sk = t$ is abduced explicitly and the consistency of $sk = t$ with other abduced facts and FEQ is checked.

The procedure which we have just described can be extended to cope with negation through the use of *negation-as-failure* [Eshghi and Kowalski, 1989]. Suppose that the selected literal of the current goal clause is *not* g. The usual negation-as-failure mechanism is used: i.e., if g cannot be proven from the theory (augmented with the current residue), then *not* g is assumed to be true. There are two sources of complications in this scheme. First, it may happen that g becomes provable later in the refutation when additional facts are abducted. To avoid this, *not* g needs to be recorded so that new clauses which are subsequently added do not violate this implicit assumption. Second, negation may be nested. Suppose there is a clause given by $g \leftarrow$ *not* h, and that h is not provable from the current residue. Then an attempt to prove *not* g using SLD-resolution with negation-as-failure (SLDNF) will fail because it is not possible to prove h. However, h might be rendered provable by adding further clauses to the residue. So rather than using SLD-resolution to try to show h, abduction is used instead and is allowed to add to the residue. This procedure can be generalized to any level of nesting, with SLD being used at even levels, and abduction at odd levels.

5.4 Query Answering in the COIN Framework

Figure 5-1 illustrates how queries are evaluated in a Context Interchange system. From a user perspective, queries and answers are couched in the relational data model: a (data-level or knowledge-level) query is formulated using a relational query language (SQL or some extension thereof), and answers can either be intensional (a mediated

query) or extensional (actual tuples satisfying the query). Examples of these queries and answers have been presented earlier in Section 3.2.

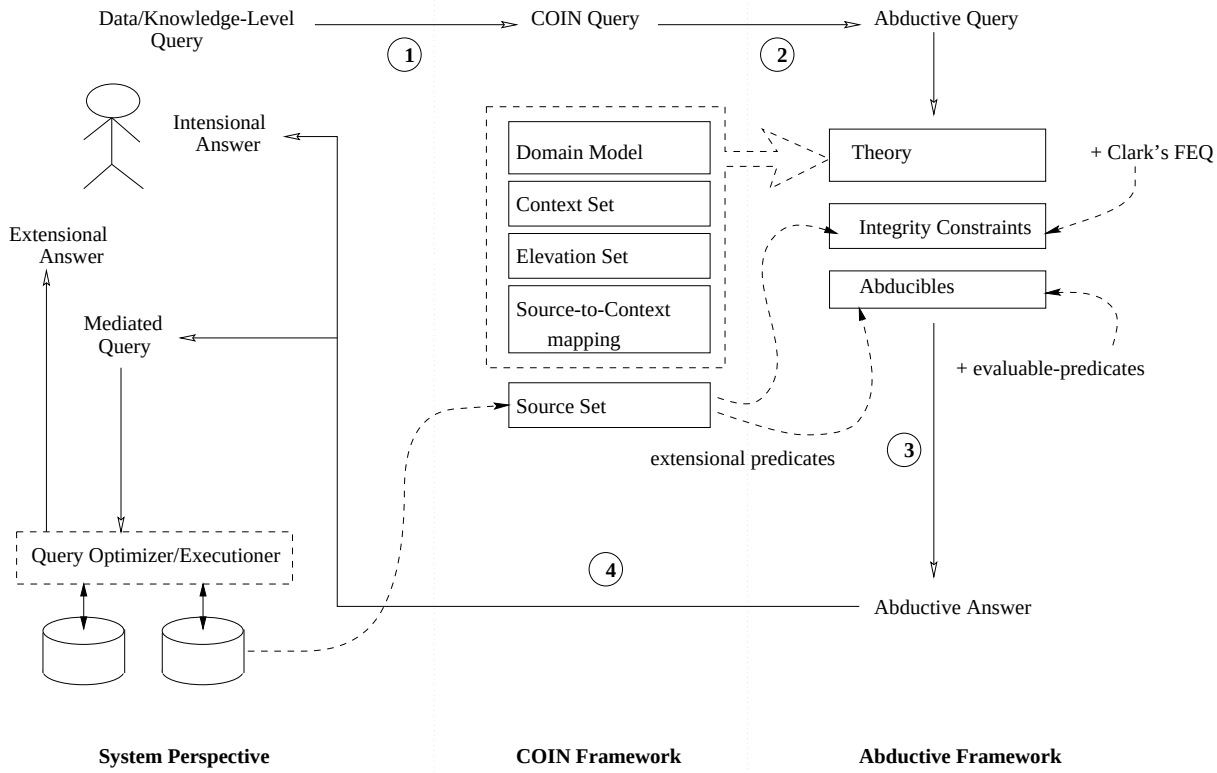

Figure 5-1: A summary of how queries are processed within the Context Interchange strategy: ① transforms a (extended) SQL query to a well-formed COIN query; ② performs the COIN to Datalog^{neg} translation; ③ is the abduction computation which generates an abductive answer corresponding to the given query; and ④ transforms the answer from clausal form back to SQL.

Transformation to the COIN Framework

Within the COIN framework, the SQL-like queries originating from users are translated to a clausal representation in the COIN language. For example, queries Q1 and Q2 in Chapter 3 can be mapped to the following clausal representations:

CQ1: $\leftarrow answer(N, R)$.

$answer(N, R) \leftarrow r_1(N, R, \_), r_2(N, E), R > E$

and correspondingly,

$$\text{CQ2: } \leftarrow \mathit{answer}(N, F_1, F_2)$$
$$\mathit{answer}(N, F_1, F_2) \leftarrow r_1(N, R, \_), R[\mathit{scaleFactor}(c_1) \rightarrow F_1],$$
$$R[\mathit{scaleFactor}(c_2) \rightarrow F_2], F_1 \neq F_2.$$

The above queries however do not capture the real intent of the user. For example, there is no recognition that "revenue" and "expenses" have different currencies and scale-factors associated with them and should not be compared "as is", that R in CQ2 is a primitive-object for which the method *scaleFactor* is not defined, or the fact that both queries originate from context c_2 which may be interpreted differently in a different context. We say that these queries are "naive", and thus must be translated to corresponding "well-formed" queries.

Definition 10 Let $<Q, c>$ be a naive query in a COIN framework $\mathcal{F}$, where c denotes the context from which the query originates. The *well-formed* query Q' corresponding to $<Q, c>$ is obtained by the following transformations:

- replace all relational operators with their "semantic" counterpart; for example, $X > Y$ is replaced with $X \overset{c}{>} Y$.
- make all relational "joins" explicit by replacing shared variables with explicit equality using the semantic-operator $\overset{c}{=}$; for example, $r_1(X, Y), r_2(X, Z)$ would be replaced with $r_1(X_1, Y), r_2(X_2, Z), X_1 \overset{c}{=} X_2$.
- similarly, make relational "selections" explicit; thus, $r_1(X, a)$ will be replaced by $r_1(X, Y)$, $Y \overset{c}{=} a$.
- replace all references to extensional relations with the corresponding semantic-relations; for example, $r_1(X, Y)$ will be replaced with $r_1'(X, Y)$.
- append to the query constructed so far, value atoms that return the value of the data elements that are requested in the query. ❑

Based on the above transformation, the well-formed query corresponding to naive queries $<\text{CQ1}, c_2>$ and $<\text{CQ2}, c_2>$, are given by

CQ1′: $\leftarrow$ *answer*(N, R).

$$\textit{answer}(N, R) \leftarrow r_1'(N_1', R', \_), r_2'(N_2', E'), N_1' \overset{c_2}{\cong} N_2', R' \overset{c_2}{\gtrsim} E',$$
$$N_1'[\textit{value}(c_2) \rightarrow N], R'[\textit{value}(c_2) \rightarrow R].$$

and

CQ2′: $\leftarrow$ *answer*(N, F_1, F_2).

$$\textit{answer}(N, F_1, F_2) \leftarrow$$
$$r1'(N', R', \_), R'[\textit{scaleFactor}(c_1) \rightarrow F_1'], R'[\textit{scaleFactor}(c_2) \rightarrow F_2'],$$
$$F_1' \overset{c_2}{\not\cong} F_2', N'[\textit{value}(c_2) \rightarrow N], F_1'[\textit{value}(c_2) \rightarrow F_1], F_2'[\textit{value}(c_2) \rightarrow F_2].$$

respectively.

Transformations to an Abductive Framework

The relationship between a COIN framework and an abduction framework can now be stated.

Definition 11 Given the COIN framework $\mathcal{F}_C = <\mathcal{S}, \mu, \mathcal{E}, \mathcal{D}, \mathcal{C}>$, this can be mapped to a corresponding abductive framework $\mathcal{F}_A$ given by $<\mathcal{T}, \mathcal{I}, \mathcal{A}>$ where

- $\mathcal{T}$ is the Datalog$^{\text{neg}}$ translation of the set of clauses given by $\mathcal{E} \cup \mathcal{D} \cup \mathcal{C} \cup \mu$;
- $\mathcal{I}$ consists of the integrity constraints defined in $\mathcal{S}$, augmented with Clark's *Free Equality Axioms* [Clark, 1978]; and
- $\mathcal{A}$ consists of the extensional predicates defined in $\mathcal{S}$, the built-in predicates corresponding to arithmetic and relational (comparison) operators, and the **system** predicate which provides the interface for system calls. ❑

Suppose $\leftarrow q(\vec{X})$ is a well-formed query in the COIN framework $\mathcal{F}_C$, the corresponding abductive framework of which is denoted by $\mathcal{F}_A = <\mathcal{T}, \mathcal{I}, \mathcal{A}>$. Without any loss of generality, we assume that $\leftarrow q(\vec{X})$ is identical in both $\mathcal{F}_C$ and $\mathcal{F}_A$. This is because Datalog$^{\text{neg}}$ is a sublanguage of COIN, and any COIN query $\leftarrow Q(\vec{X})$ can always be

transformed to a Datalog$^{\text{neg}}$ query $\leftarrow q(\vec{X})$ by adding the Datalog$^{\text{neg}}$-translation of the clause $q(\vec{X}) \leftarrow Q(\vec{X})$ into the theory $\mathcal{T}$.

Given an abductive framework $<\mathcal{T}, \mathcal{I}, \mathcal{A}>$, and the query $\exists\vec{X}\, q(\vec{X})$. Suppose $\Delta = \{p_1, \ldots, p_m\}$ is an abductive answer for $q(\vec{X})\theta$, then it follows that

$$\mathcal{T} \models (\, q(\vec{X})\theta \leftarrow p_1, \ldots, p_m \,)$$

This result follows from the fact that p_i's are ground for $i = 1, \ldots, m$, so a set of ground atoms in fact represents their conjunction. The conjunct $p_1 \wedge \cdots \wedge p_m$ constitutes a precondition for $q(\vec{X})\theta$. Suppose $K = \{sk_0, \ldots\}$ is the set of Skolem constants introduced by the abduction step, and φ is a "reverse" substitution $\{sk_i/Y_i\}$ where $sk_i \in K$ and Y_i is a distinct variable not in $\vec{X}$. Then, we say that the tuple $(\, \exists\vec{Y}\, (p_1, \ldots, p_m)\varphi, \theta\varphi \,)$ is an *intensional answer* for the query $\exists\vec{X}\, q(\vec{X})(\theta\varphi)$. This fact is not surprising given that Motro and Yuan [Motro and Yuan, 1990] suggested that intensional answers can be obtained from the "dead-ends" of "derivation trees" corresponding to a query. Although it was not recognized as such, the procedure described in [Motro and Yuan, 1990] is in fact a naive implementation of SLD+Abduction (without any consistency checking). From the perspective of the user issuing a naive query, the intensional answer can also be interpreted as the corresponding *mediated answer*.

An an illustration of the preceding comments, the evaluation of CQ2$'$ in the abductive framework yields the following abductive answer:

$$\Delta = \{r_1(sk_0, sk_1, sk_2), sk_2 = \texttt{'JPY'}\}, \theta = \{N/sk_0, F_1/\texttt{1\,000}, F_2/\texttt{1}\}$$

The reverse substitution φ is given by $\{sk_0/Y_0, sk_1/Y_1, sk_2/Y_2\}$, and thus the intensional answer (equivalently, the mediated query) is:

$$(\, \exists Y_0, Y_1, Y_2\, (r_1(Y_0, Y_1, Y_2), Y_2 = \texttt{'JPY'}),\ \{N/Y_0, F_1/\texttt{1\,000}, F_2/\texttt{1}\}\ \,)$$

which translates to MQ2 shown in Chapter 3. If $\{Y_0/\texttt{'NTT'}, Y_1/\texttt{1\,000\,000}, Y_2/\texttt{'JPY'}\}$ is an answer for the above mediated query, then the answer for the original user query is given by $\{N/\texttt{'NTT'}, F_1/\texttt{1\,000}, F_2/\texttt{1}\}$.

5.5 Illustrative Example

In this section, we provide an example illustrative of the computation involved in query mediation (equivalently, obtaining the intensional answer to a query).

Consider the query Q3 (a simplified variant of Q2) which is issued from context c_1, which queries relation r_1 for the scale-factors of revenues in context c_1:

```
Q3:  SELECT r1.cname, r1.revenue.scaleFactor IN c1
     FROM r1;
```

The (well-formed) clausal representation for this query is given by

$$\text{CQ3:} \quad \leftarrow answer(N, F).$$
$$answer(N, F) \leftarrow r1'(N', R', \_), N'[value(c_1) \rightarrow N], R'[scaleFactor(c_1) \rightarrow F'], F'[value(c_1) \rightarrow F].$$

Figure 5-2 shows one possible refutation of this query using the SLD+Abduction algorithm described earlier. For better clarity, the refutation is shown using COIN clauses rather than Datalog. The clauses used for resolving the goal clauses are those shown earlier in Figure 4-1, 4-2 and 4-3.

To aid in appreciating the chain of reasoning, we offer the following highlights on the refutation:

- The refutation begins with the query as given, with Δ initialized to the empty set.
- At step (3), the literal $r_1(N, \_, \_)$ cannot be further resolved. Since r_1 is an extensional predicate (and hence abducible), it is removed from the goal clause and its Skolemized form, $r_1(sk_0, sk_1, sk_2)$, is added to Δ.
- At step (6), the literal $scaleFactor(c_1, f_{\text{r1\#revenue}}(sk_0))[value(c_1) \rightarrow F]$ can be resolved with two different clauses (where $F = 1$ and $F = 1\,000$). One is chosen arbitrarily (in this case, $F = 1$); the other will be selected on backtracking and will eventually lead to another refutation.

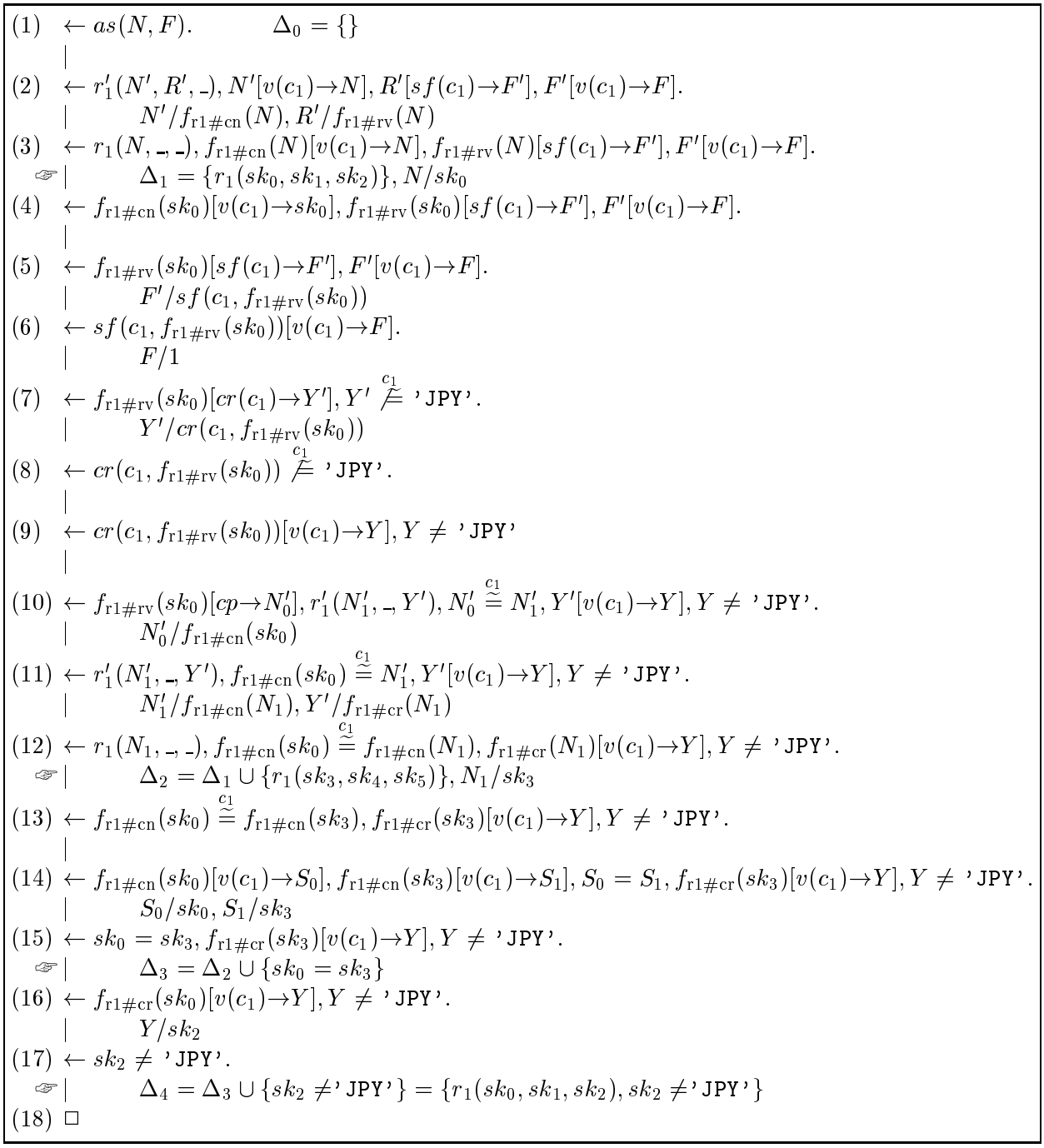

Figure 5-2: One possible refutation for query CQ3. Method and functor names are abbreviated where possible (e.g., *cr* = *currency*). The resolution step labeled ☞ is where a literal is abducted. The abductive answer corresponding to this refutation is given by Δ_4, and the intensional answer by $(\Delta_4, \{N/sk_0, F/1\})$.

- To arrive at a successful refutation, the currency for the revenue-object at hand must not be `'JPY'` when evaluated in context c_1 (see step (8)). To determine if this is the case, it is necessary to identify the currency value from the extensional relation r_1 (see corresponding axiom for assigning currency values in Figure 4-3). This eventually leads to the expansion of the goal clause as shown in step (10).

- In step (12), the extensional relation is referenced again. In the absence of other information, we are not allowed to assume that it is the same "fact" which has been abducted: i.e., we will need to add a new Skolemized fact, $r_1(sk_3, sk_4, sk_5)$ to Δ.

- In step (15), the equality constraint on the objects $f_{\text{r1\#cname}}(sk_0)$ and $f_{\text{r1\#cname}}(sk_3)$ leads to the constraint $sk_0 = sk_3$. Since '=' is abducible (it is an evaluable predicate), it is added to Δ. At this point, the functional dependency `cname` $\rightarrow$ {`revenue, currency`} generates further the constraints $sk_1 = sk_4$ and $sk_2 = sk_5$, which in turn allow us to merge the two facts $r_1(sk_0, sk_1, sk_2)$ and $r_1(sk_3, sk_4, sk_5)$.

- Finally, in step (17), the literal sk_2 = `'JPY'` is abducted, which leads to a refutation. The abductive answer corresponding to this refutation is given by $\Delta = \{r_1(sk_0, sk_1, sk_2), sk_2 =$ `'JPY'`$\}$. The substitution, restricted to variables $\{N, F\}$, is given by $\{N/sk_0, F/1\}$.

This intensional answer, translated to SQL, is given by:

```
SELECT r1.cname, 1 FROM r1 WHERE r1.currency <> 'JPY';
```

On backtracking, the other solution corresponding to $F =$`1 000` will be obtained. The complete answer returned to the user is thus given by:

```
MQ3:  SELECT r1.cname, 1 FROM r1 WHERE r1.currency <> 'JPY'
      UNION
      SELECT r1.cname, 1 000 FROM r1 WHERE r1.currency = 'JPY';
```

The correspondences between integrity checking and semantic query optimization can be clearly seen in the above example. At step (15), the functional dependencies r_1 allows the initial constraint $(sk_0 = sk_3)$ to be propagated and eventually allow $r_1(sk_3, sk_4, sk_5)$ to be eliminated from the abductive answer. If it were not so, the intensional answer obtained would instead be:

```
SELECT rel1.cname, 1 FROM r1 rel1, r1 rel2
WHERE rel1.cname = rel2.cname;
```

which would include a redundant second reference to r_1. This second answer is unintuitive, and obviously would lead to suboptimal performance if executed without further optimization. In the more general scenario, constraints can be useful in pruning an entire refutation altogether. For instance, if Q3 had been:

```
Q3':  SELECT r1.cname IN c1, r1.revenue.scaleFactor IN c1
      FROM r1 WHERE r1.currency = 'JPY';
```

we will eventually end up trying to abduct $sk_2 =$`'JPY'` where $sk_2 \neq$ `'JPY'` is already present in Δ, thus resulting in an unsuccessful refutation. In this case, the mediated query MQ3$'$ will consist of only the second select-statement in MQ3.

Chapter 6

The Context Interchange Prototype

The goal of the Context Interchange Prototype[1] is to provide a demonstration of the feasibility and features of Context Interchange strategy. In commensuration with the goals of this Thesis, we are particularly interested in a concrete implementation of the Context Mediator. In Section 6.1 , we present the architecture of this Prototype and a brief description of its components. Section 6.2 contains an in-depth discussion of the implementation of the Context Mediator, which is responsible for transforming a user query to a mediated query. The last section concludes this chapter by portraying how a user might make use of the system to gain "mediated access" to disparate information sources on the World Wide Web.

[1] A number of students (past and present) have contributed to the current implementation. In no particular order, these include Adil Daruwala, Kofi Fynn, Karim Hussein, Marta Jakobisiak, Thomas Lee, Andy Loh, Tito Pena, and Jessica Qu. Of course, Stuart Madnick and Michael Siegel played the key role of sitting on top of everybody. At the time of writing, queries are processed using an underlying Oracle system, which is sufficient for the purpose of demonstrating how queries are mediated. Stéphan Bressan is currently leading the effort in constructing a query optimizer/executioner that will allow greater latitude for experimenting with various query optimization strategies.

6.1 The Context Interchange Prototype: Overview

Figure 6-1 shows the architecture of the Prototype which is being implemented, which consists of three distinct groups of processes.

- *Client Processes* provide the interaction with receivers and route all database requests to the Context Mediator. An example of a client process is the *multi-database browser* [Jakobisiak, 1996], which provides a point-and-click interface for formulating queries to multiple sources, and for displaying the answers obtained. More generally, any application program which issues queries to one or more sources can be considered a client process. For example, Microsoft Excel has the capability of issuing an SQL-query encapsulated as an ODBC-request. In the current implementation, this request can be intercepted by a custom ODBC driver, which redirects the request to the Context Mediator.

- *Server Processes* refer to *database gateways* and *wrappers*. Database gateways provide physical connectivity to databases on a network: the goal is to insulate the Mediator Processes from the idiosyncrasies of different database management systems by providing a uniform protocol for database access as well as a canonical query language (and data model) for the formulating the queries. Wrappers, on the other hand, provide richer functionalities by allowing semi-structured documents on the World-Wide-Web to be queried as if these were regular databases. This is accomplished by defining an *export schema* for each of these web-sites, and describing how attribute-values can be extracted from the web-pages using regular-expressions [Qu, 1996].

- *Mediator Processes* refer to the system components which collectively provide the mediation services: these include the Context Mediator (which rewrites a user-query to a mediated query), the Optimizer (which produces an optimal query evaluation plan based on the mediated query), and the Executioner (which executes the plan by dispatching subqueries to the Server Processes, collating and operating on the intermediary results, and returning the final answer to the

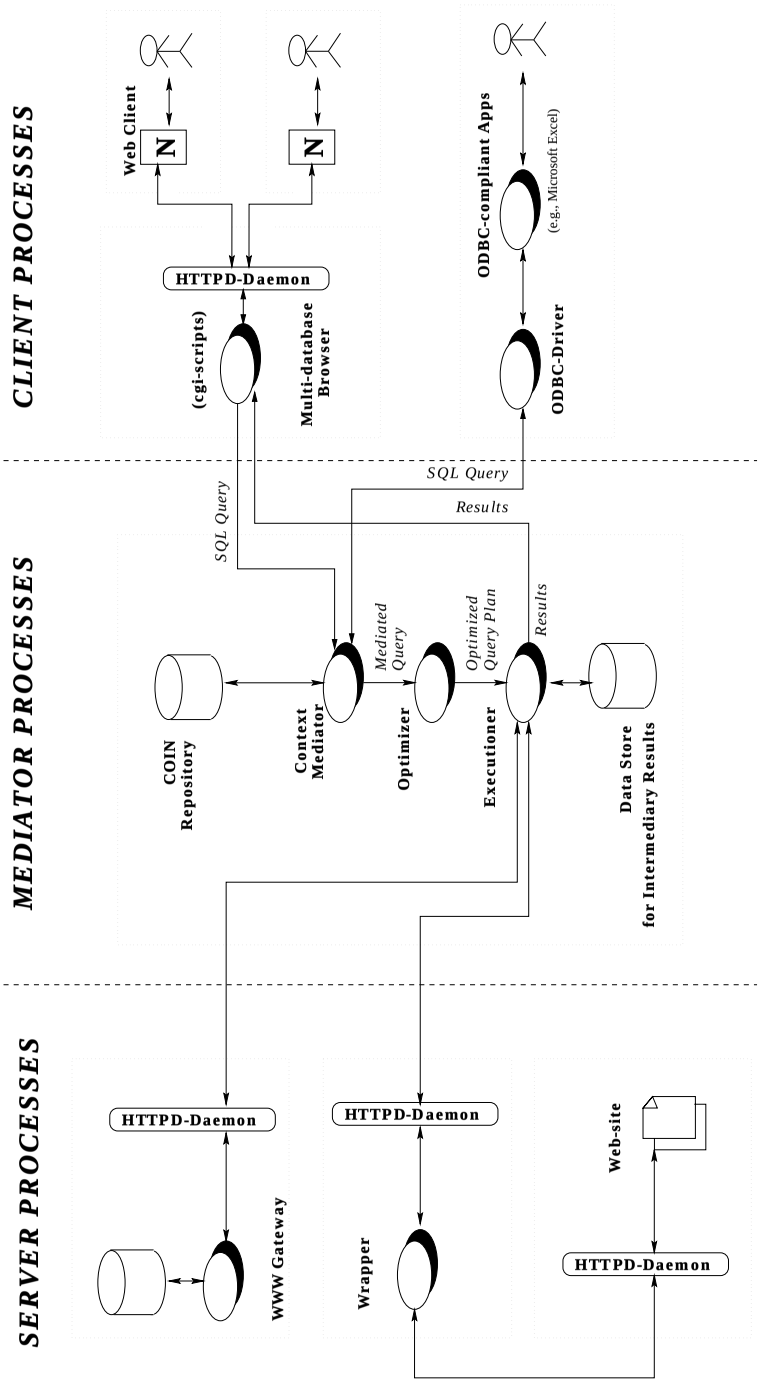

Figure 6-1: Architectural overview of the Context Interchange Prototype.

Client Process).

The Mediator Processes are supported by two repositories. The COIN *Repository* functions as a registry for knowledge pertaining to the integration task at hand. This include the axioms which make up the COIN framework and the Export Schema corresponding to each source. The second repository serves as a temporary data store for the Executioner. In the current implementation, this takes the form of a Oracle database, allowing intermediate results to be stored and operated upon using facilities of the Oracle DBMS.

In line with our goals of developing a prototype which is easily accessible (outside the confines of our research laboratory) as well as to allow us to focus on the mediation technology which is of most interest to us, we have chosen to leverage on the infrastructure of the World-Wide-Web (WWW) whenever possible. For example, we rely on the *Internet Protocol (IP)* to provide connectivity across heterogeneous networks and hardware platform, the *Hypertext Transfer Protocol (HTTP)* protocol for communication across different gateways, *Universal Resource Locators (URLs)* as a universal addressing scheme for identifying and locating resources (in particular, information sources) on the WWW, and *Hypertext Mark-up Language (HTML)* for displaying the query answers.

Constructing a Prototype leveraging on these protocols brings about a number of benefits. First, this allows us to develop programs which are highly portable. For instance, most of the client and server processes (including the multidatabase browser, database gateways, and wrappers) are implemented as cgi-scripts written in a scripting language called Perl. These programs can be executed on virtually any hardware and software platform, ranging from high-end Unix workstations to desktop personal computers. Second, we also have the option of distributing different processes transparently across different systems, both for load-balancing and also out of respect for the autonomy of different systems. For instance, it is not necessary for users to install any COIN-specific applications on their system prior to accessing the multidatabase browser. Instead, the latter can be executed on a remote site using the ubiquitous Web Browser (e.g., the Netscape client). In the same way, wrappers can

be executed on a different system from the actual web-site furnishing the documents. This turns out to be a critical feature since we typically do not have the authority to execute COIN-specific programs on these sites.

6.2 Implementation of the Context Mediator

The Context Mediator is implemented in *ECLiPSe*[2], which is an efficient and robust Prolog implementation distributed by the ECRC. In actuality, the Context Mediator consists of four distinct components which are loosely-connected to one another as shown in Figure 6-2. At the heart of the Context Mediator is the *Abduction Engine* which implements the extended SLD+Abduction algorithm as was described in Chapter 5. Since computation of the abductive answer is performed within a Horn-clause (HC) framework, we need to translate both the user-query as well as COIN clauses to statements in Datalog^{neg}, and on obtaining the answer, perform the reverse translation to SQL. In the absence of aggregation operators, the SQL-to-HC and HC-to-SQL compilers are relatively straight-forward since both of these languages shares a common grounding in predicate calculus.

The Abduction Engine

The Abduction Engine takes the form of a *meta-interpreter* [Sterling and Shapiro, 1994, Chapter 17], the skeleton of which is shown in Listing 6.2.1.

We offer the following declarative reading of the meta-interpreter program:

- clauses (1), (2), and (7) corresponds to the vanilla meta-interpreter in the Prolog folklore. Clause (1) states that the empty goal, represented by the constant `true`, is true. Clause (2) states that a conjunction `(A,B)` is true if `A` is true and `B` is true. Clause (7) performs the resolution step by unifying a goal literal with a clause in the "program". It is also responsible for giving different solutions on backtracking (by performing resolution on a different clause in the program).

[2]ECLiPSe: The ECRC Constraint Logic Parallel System. More information can be obtained at `http://www.ecrc.de/eclipse/`.

Listing 6.2.1 Skeleton of the meta-interpreter implementing the Abduction Engine.

```
abductively_solve(true, A0, A0, C0, C0)  :- !.   % (1)
abductively_solve((H,T), A0, A2, C0, C2) :- !,   % (2)
  abductively_solve(H,A0,A1,C0,C1),
  abductively_solve(T,A1,A2,C1,C2).
abductively_solve(prolog(Lit),A0,A0,C0,C0) :- !, % (3)
  Lit.
abductively_solve(X=Y,A0,A0,C0,C1) :- !,         % (4)
  (  consistency_check(X=Y,A0,C0) ->
        insert_constraint(X=Y,C0,C1).
     ;
        fail
  ).
abductively_solve(not X=Y,A0,A0,C0,C1) :- !,     % (5)
  (  consistency_check(not X=Y,A0,C0) ->
        insert_constraint(not X=Y,C0,C1).
     ;
        fail
  ).
abductively_solve(L,A0,A1,C0,C0) :-              % (6)
  abducible(Lit), !,
  (
     consistency_check(Lit,A0,C0) ->
        abduct(Lit,A0,A1),
     ;
        fail
  ).
abductively_prove(Lit,A0,A1,C0,C1) :-            % (7)
  (
     consistency_check(Lit,A0) ->
       (  clause(Lit,Body),
          abductively_prove(Body,A0,A1,C0,C1)
       )
     ;
       fail
  ).
```

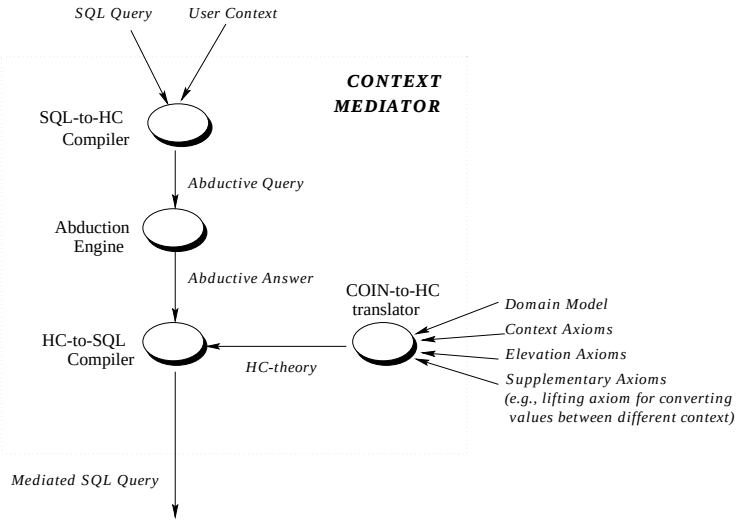

Figure 6-2: Context Mediator Internals.

- clause (3) provides an escape mechanism into the underlying prolog system which is handy for manipulation of high-order terms. For example, we use `prolog(=..(X,[H,L]))` for synthesizing and taking apart Skolem terms. The goal literal `prolog(Lit)` evaluates to true if the execution of `Lit` succeeds, and false otherwise.

- clauses (4) and (5) provide for separate treatment of equality and disequality constraints. Specifically, whenever an equality or disequality constraint is encountered, the new constraint is tested against existing constraints. If there is a contradiction (e.g., when the same Skolem constant is simultaneously `='USD'` and `<>'USD'`), `consistency_check` fails and the meta-interpreter backtracks to the next (`abductively_solve`) clause. Otherwise, the new constraint is added to the underlying collection of constraints with the appropriate propagations.

- finally, clause (6) determines if a goal literal is abducible, and if so, update the abductive answer (`Ans`) with the newly abducted literal. In our framework, a literal is abducible if it corresponds to an extensional (database) relation, or if it corresponds to external programs (for example, an executable program responsible for realizing some conversion function). Abducibles are declared

in the program through the use of the predicate `abducible_predicate`; for example,

```
abducible(L) :- prolog(functor(L,Functor,Arity)),
                abducible_predicate(Functor,Arity).
abducible_predicate(is,2).
abducible_predicate(r1,3).
abducible_predicate(r2,2).
```

All extensional database predicates are declared to be `abducible_predicate`s in the Enriched Schema. The declaration of `is/2` as abducible allows arithmetic expressions in general to be part of the abductive answer.

Notice that in steps (4) to (7), integrity checking is performed at each step to ensure that the answers obtained is consistent with known integrity constraints. In the current implementation, we have made explicit provisions for *key constraints*. Hence, if the same extensional relation is to be abducted twice with the key bound to the same value, but having distinct values bound to the non-prime attributes, this will be signal as a constraint violation, causing the meta-interpreter to backtrack.

The Abductive Query

The abductive procedure is initiated by invoking `abductively_solve` as follows:

```
:- abductively_solve(Goal,[],Ans,[],Constr).
```

where `Goal` corresponds to the abductive query, `Ans` is a list of literals which have been abducted, and `Constr` is a set of (equality and disequality) constraints on the abductive answer. Each successful invocation of `abductively_solve` returns one answer corresponding to one conjunctive query in the mediated query. On backtracking, a different abductive answer will be obtained. The set of all abductive answers comprises the mediated query (which takes the form of a disjunction of conjunctive queries).

As an illustration, consider the following abductive query which corresponds to the query Q1 introduced in Chapter 3:

```
answer(C,R) :- r1_p(E1,X,_), r2_p(E2,Y),
               sem_op(=,c2,E1,E2), sem_op(>,c2,X,Y),
               value(E1,c2,C), value(X,c2,R).
```

The abductive answer is obtained by asking the query

```
:- abductively_prove(answer(C,R),[],Ans,[],Constr).
```

With respect to the example in Chapter 3, this returns with the following bindings:

```
C = X, R = Y,
Ans = [r1(X,Y,U), r2(X,Z), Y>Z], Constr = [U='USD']
```

On successive backtracking, we obtain the other two answers given by

```
C = X, R = Y1,
Ans = [r1(X,Y,U), r2(X,Z), r3(U,V,T), Y0 is Y*1000, Y1 is Y0*T, Y1>Z],
Constr = [U='JPY', V='USD']
```

and

```
C = X, R = Y1,
Ans = [r1(X,Y,U), r2(X,Z), r3(U,V,T), Y1 is Y*T, Y1>Z],
Constr = [U<>'JPY', U<>'USD', V='USD']
```

These abductive answers transform directly to the mediated query MQ1 shown earlier in Chapter 3.

6.3 Mediated Data Access: A User's Perspective

As a concrete illustration of how mediated data access takes place using the Prototype, we present a brief description of a typical interaction session as seen from a user's perspective.

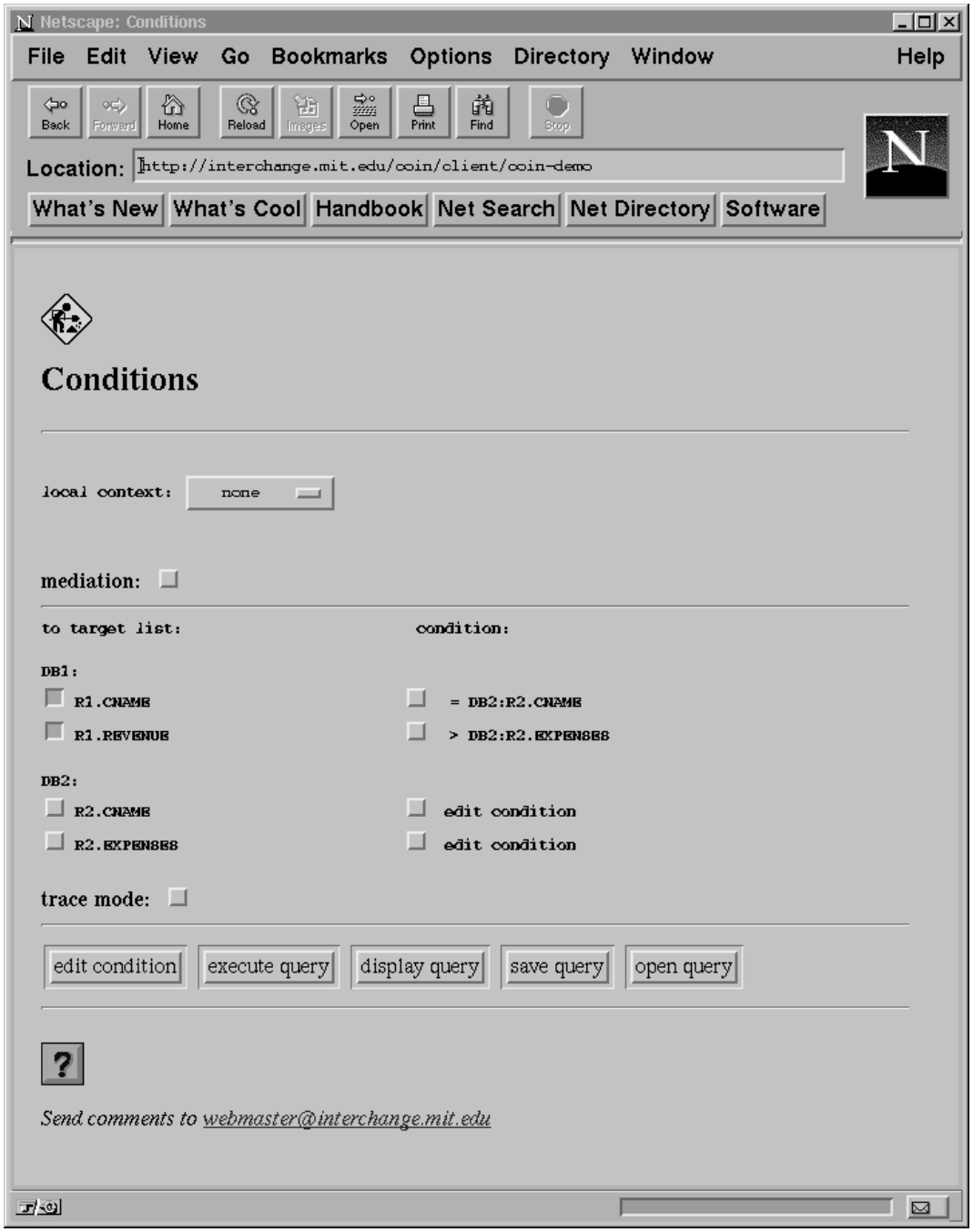

Figure 6-3: Screen-shot of the multidatabase browser.

Upon invocation, the multidatabase browser offers the user a list consisting of all the sources that are accessible. Having made the choice, the user will be prompted to identify the relations and the attributes in those relations which are needed to formulate a query. The query is further constrained by associating conditions with each attribute selected. The screen-shot depicted in Figure 6-3 is taken the this point, where the query is fully specified. At this juncture, the user is ready to submit the query for mediation and/or execution. The user could choose to by-pass the Context Mediator altogether by *not* turning off the check-box labeled "mediation": in this case, the query will be executed as a regular multi-database query without any attempts to mediate potential conflicts. If, however, mediation is turned on, the user must specify a context within which the mediation takes place. This is accomplished by choosing from among a number of predefined contexts available through a pick-list at the top of the page. As an aid to understanding the underlying mediation process, there is a check-box labeled "trace mode": if checked, this allows the user to navigate through several intermediary screens which report the transformations underlying the mediation process, before presenting the answers to the query.

Chapter 7

Conclusion

We have presented in this Thesis a tightly-woven tapestry of ideas derived from different threads in the literature in artificial intelligence (on "contexts"), databases (on "heterogeneous databases" and "semantic query optimization"), logic programming (on "abductive logic programming" and "meta-logic"), and others which are already present at the confluence of different scholarly traditions (e.g., "deductive object-oriented data models" and "intensional answers"). The various results and insights integrated together in a formal framework for the Context Interchange strategy, and provide a well-founded basis for representing and reasoning about data semantics in disparate sources and receivers. Specifically, we have described how data semantics in disparate systems can be articulated using a "object-logic", and how logical inferences (in particular, abduction) can be used to provide mediated access to both data and data-semantics. At the same time, we showed that the COIN framework presents a viable alternative to classical and contemporary integration approaches by by allowing different kinds of information to be more easily accessed, by making possible the sustenance of an infrastructure that mitigates the complexity in the creation and maintenance of large-scale systems, and by isolating changes in different components which are only loosely-coupled together.

7.1 Future Work

There are many interesting issues which we are only beginning to explore. We mention below two of these undertakings.

As is pointed in [Lu et al., 1992], the autonomy and heterogeneity of sources present new challenges for query processing and optimization which are not the same as those in distributed database systems. These differences stem from constraints which are characteristic of the underlying environment; for example, different sources may differ in their query-handling ability, cost models may not be known, and data conversions may incur large hidden costs which are not accounted for previously. As we have shown earlier, the detection of unsatisfiable answers in the abductive framework constitute a form of semantic query optimization which presents huge payoffs. We are currently examining how we can take advantage of this framework to deal with more general constraints for identifying queries which can be executed more efficiently. To this end, we have been able to make use of the existing prototype as a testbed on which theoretical insights can be rapidly implemented and experimented with.

The richness of the representational formalism is a two-edged sword since it presents also greater scope for abuse. While it is unlikely that there will ever be a "definitive guide" to context modeling, case studies, evaluation criteria, prescriptive guidelines, and tools are in dire need. At this moment, we are working with several industry information-providers in applying this mediation technology to the "real world" problems encountered by them. We are hopeful that these experiences will be instrumental in developing and validating integration methodologies that are grounded in practice.

Bibliography

Abiteboul, S., Lausen, G., Uphoff, H., and Walker, E. (1993). Methods and rules. In *Proceedings of the ACM SIGMOD Conference*, pages 32–41, Washington, DC.

Agarwal, S., Keller, A. M., Wiederhold, G., and Saraswat, K. (1995). Flexible relation: An approach for integrating data from multiple, possibly inconsistent databases. In *Proc. IEEE Intl Conf on Data Engineering*, Taipei, Taiwan.

Ahmed, R., Smedt, P. D., Du, W., Kent, W., Ketabchi, M. A., Litwin, W. A., Raffi, A., and Shan, M.-C. (1991). The Pegasus heterogeneous multidatabase system. *IEEE Computer*, 24(12):19–27.

Aït-Kaci, H. and Nasr, R. (1986). LOGIN: A logic programming language with built-in inheritance. *Journal of Logic Programming*, 3(3):185–215.

Arens, Y. and Knoblock, C. A. (1992). Planning and reformulating queries for semantically-modeled multidatabase systems. In *Proceedings of the 1st International Conference on Information and Knowledge Management*, pages 92–101.

Atkinson, M. P., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D., and Zdonik, S. (1989). The object-oriented database system manifesto (invited paper). In *Proc. First Int'l. Conf. on Deductive and Object-Oriented Databases*, Kyoto, Japan.

Bancilhon, F., Delobel, C., and Kanellakis, P., editors (1992). *Building an Object-Oriented Database System: The Story of O_2*. Morgan Kaufmann Publishers, Inc.

Bancilhon, F. and Ramakrishnan, R. (1986). An amateur's introduction to recursive query processing strategies. In *Proc. ACM SIGMOD*, pages 16–52.

Borgi, A., Mancarella, P., Pedreschi, D., and Turini, F. (1990). Compositional operators for logic theories. In Lloyd [1990], pages 117–134.

Bouguettaya, A. and King, R. (1992). Large multidatabases: issues and directions. In Hsiao, D. K., Neuhold, E. J., and Sacks-Davis, R., editors, *Proceedings of the IFIP WG2.6 Database Semantics Conference on Interoperable Database Systems (DS-5)*, pages 55–68, Lorne, Victoria, Australis. North-Holland.

Breitbart, Y. J. and Tieman, L. R. (1985). ADDS: Heterogeneous distributed database system. In Schreiber, F. and Litwin, W., editors, *Distributed Data Sharing Systems*, pages 7–24. North Holland Publishing Co.

Bright, M., Hurson, A., and Pakzad, S. (1992). A taxonomy and current issues in multidatabase systems. *IEEE Computer*, 25(3):50–60.

Brogi, A. and Turini, F. (1991). Metalogic for knowledge representation. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR-91)*, pages 61–69, Cambridge, MA.

Buvač, S. (1995). Quantificational logic of context. In *Proceedings of the Workshop on Modeling Context in Knowledge Representation and Reasoning (held at the Thirteenth International Joint Conference on Artificial Intelligence).*

Ceri, S., Gottlob, G., and Tanca, L. (1990). *Logic Programming and Databases.* Springer-Verlag, Berlin, Germany.

Chakravarthy, U. S., Grant, J., and Minker, J. (1990). Logic-based approach to semantic query optimization. *ACM Transactions on Database Systems*, 15(2):162–207.

Chang, C.-L. and Lee, R. C.-T. (1973). *Symbolic Logic and Mechanical Theorem Proving.* Academic Press.

Chen, W. and Warren, D. S. (1989). C-logic for complex objects. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 369–378.

Clark, K. (1978). Negation as failure. In Gallaire, H. and Minker, J., editors, *Logic and Data Bases*, pages 292–322. Plenum Press.

Collet, C., Huhns, M. N., and Shen, W.-M. (1991). Resource integration using a large knowledge base in Carnot. *IEEE Computer*, 24(12):55–63.

Console, L., Theseider Dupre, D., and Torasso, P. (1991). On the relationship between abduction and deduction. *Journal of Logic and Computation*, 1(5):661–690.

Cox, P. T. and Pietrzykowski, T. (1986). Causes for events: their computation and applications. In Siekmann, J. H., editor, *Proceedings CADE-86*, pages 608–621. Springer-Verlag.

Daruwala, A., Goh, C. H., Hofmeister, S., Hussein, K., Madnick, S., and Siegel, M. (1995). The context interchange network prototype. In *Proc of the IFIP WG2.6 Sixth Working Conference on Database Semantics (DS-6)*, Atlanta, GA. To appear in LNCS (Springer-Verlag).

Dayal, U. and Hwang, H.-Y. (1984). View definition and generalization for database integration in a multidatabase system. *IEEE Software Engineering*, 10(6):628–645.

Denecker, M. (1993). *Knowledge Representation and Reasoning in Incomplete Logic Programming*. PhD thesis, Departement Computerwetenschappen, Katholieke Universiteit Leuven, Belgium.

Denecker, M. and Schreye, D. D. (1992a). On the duality of abduction and model generation. In *Proc Int Conf on Fifth Generation Computer Systems*, pages 650–657.

Denecker, M. and Schreye, D. D. (1992b). SLDNFA: an abductive procedure for normal abductive programs. In *Proc. Int Conf and Symposium on Logic Programming*. 686–700.

Dobbie, G. and Topor, R. (1995). On the declarative and procedural semantics of deductive object-oriented systems. *Journal of Intelligent Information Systems*, 4:193–219.

Eiter, T. and Gottlob, G. (1993). The complexity of logic-based abduction. In Enjalbert, P., Finkel, A., and Wagner, K. W., editors, *Proc. 10th Symposium on Theoretical Aspects of Computing (STACS-93)*, pages 70–79. Extended paper to appear in *Journal of the ACM*.

Eshghi, K. (1988). Abductive planning with event calculus. In *Proc. 5th International Conference on Logic Programming*, pages 562–579.

Eshghi, K. (1993). A tractable set of abduction problems. In *Proc. 13th International Joint Conference on Artificial Intelligence*, pages 3–8, Chambery, France.

Eshghi, K. and Kowalski, R. A. (1989). Abduction compared with negation by failure. In *Proc. 6th International Conference on Logic Programming*, pages 234–255, Lisbon.

Faquhar, A., Dappert, A., Fikes, R., and Pratt, W. (1995). Integrating information sources using context logic. In *AAAI-95 Spring Symposium on Information Gathering from Distributed Heterogeneous Environments*. To appear.

Finger, J. S. and Genesereth, M. R. (1985). Residue: A deductive approach to design synthesis. Technical Report TR-CS-8, Stanford University.

Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J., and Widom, J. (1995). The TSIMMIS approach to mediation: data models and languages. In *Proc NGITS (Next Generation Information Technologies and Systems)*, Naharia, Israel. To appear.

Garcia-Solaco, M., Saltor, F., and Castellanos, M. (1996). Semantic heterogeneity in multidatabase systems. In Bukhres, O. A. and Elmagarmid, A. K., editors, *Object-Oriented Multidatabase Systems: A Solution for Advanced Applications*, chapter 5, pages 129–202. Prentice-Hall.

Goh, C. H., Madnick, S. E., and Siegel, M. D. (1994). Context interchange: overcoming the challenges of large-scale interoperable database systems in a dynamic

environment. In *Proceedings of the Third International Conference on Information and Knowledge Management*, pages 337–346, Gaithersburg, MD.

Gruber, T. R. (1991). The role of common ontology in achieving sharable, reusable knowledge bases. In Allen, J. A., Files, R., and Sandewall, E., editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 2nd International Conference*, pages 601–602, Cambridge, MA. Morgan Kaufmann.

Guha, R. V. (1991). Contexts: a formalization and some applications. Technical Report STAN-CS-91-1399-Thesis, Department of Computer Science, Stanford University.

Hull, R. and King, R. (1987). Semantic database modeling: survey, applications, and research issues. *ACM Computing Surveys*, 19(3):201–260.

Hurson, A. R., Bright, M. W., and Pakzad, S. H. (1994). *Multidatabase Systems: an advanced solution for global information sharing*. IEEE Computer Society Press.

Imielinski, T. (1987). Intelligent query answering in rule based systems. *Journal of Logic Programming*, 4(3):229–257.

Jakobisiak, M. (1996). Programming the web – design and implementation of a multidatabase browser. Technical Report CISL WP#96-04, Sloan School of Management, Massachusetts Institute of Technology.

Jonker, W. and Schütz, H. (1995). The ECRC multidatabase system. In *Proc ACM SIGMOD*, page 490.

Kakas, A. C., Kowalski, R. A., and Toni, F. (1993). Abductive logic programming. *Journal of Logic and Computation*, 2(6):719–770.

Kakas, A. C. and Mancarella, P. (1990a). Database updates through abduction. In *Proceedings 16th International Conference on Very Large Databases*, pages 650–661, Brisbane, Australia.

Kakas, A. C. and Mancarella, P. (1990b). Generalised stable models: a semantics for abduction. In *Proc. 9th European Conference on Artificial Intelligence (ECAI-90)*, pages 385–391, Stockolm.

Kifer, M. and Lausen, G. (1989). F-logic: a higher-order language for reasoning about objects, inheritence and scheme. In *Proc ACM SIGMOD*, pages 134–146.

Kifer, M., Lausen, G., and Wu, J. (1995). Logical foundations of object-oriented and frame-based languages. *JACM*, 4:741–843.

Kifer, M. and Wu, J. (1989). A logic for object-oriented logic programming (Maier's O-logic revisited). In *Proc ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 373–393.

Kim, W. and Seo, J. (1991). Classifying schematic and data heterogeneity in multidatabase systems. *IEEE Computer*, 24(12):12–18.

Kowalski, R. A. (1990). Problems and promises of computational logic. In Lloyd [1990], pages 1–36.

Kowalski, R. A. (1991). Logic programming in artificial intelligence. In *Proc. of the IJCAI-91*.

Krishnamurthy, R., Litwin, W., and Kent, W. (1991). Language features for interoperability of databases with schematic discrepancies. In *Proceedings of the ACM SIGMOD Conference*, pages 40–49.

Kuhn, E. and Ludwig, T. (1988). VIP-MDBMS: A logic multidatabase system. In *Proc Int'l Symp. on Databases in Parallel and Distributed Systems*.

Landers, T. and Rosenberg, R. (1982). An overview of Multibase. In *Proceedings 2nd International Symposium for Distributed Databases*, pages 153–183.

Lenat, D. B. and Guha, R. (1989). *Building large knowledge-based systems: representation and inference in the Cyc project*. Addison-Wesley Publishing Co., Inc.

Litwin, W. (1992). O*SQL: A language for object oriented multidatabase interoperability. In Hsiao, D. K., Neuhold, E. J., and Sacks-Davis, R., editors, *Proceedings of the IFIP WG2.6 Database Semantics Conference on Interoperable Database Systems (DS-5)*, pages 119–138, Lorne, Victoria, Australis. North-Holland.

Litwin, W. and Abdellatif, A. (1987). An overview of the multi-database manipulation language MDSL. *Proceedings of the IEEE*, 75(5):621–632.

Lloyd, J. W. (1987). *Foundations of logic programming.* Springer-Verlag, 2nd, extended edition.

Lloyd, J. W., editor (1990). *Computational Logic: Symposium Proceedings*, Brussels. Springer-Verlag.

Lu, H., Ooi, B.-C., and Goh, C. H. (1992). On global multidatabase query optimization. *ACM SIGMOD Record*, 20(4):6–11.

Maier, D. (1986). A logic for objects. Technical Report CS/E-86-012, Oregon Graduate Center, Beaverton, OR.

Malone, T. W. and Rockart, J. F. (1991). Computers, networks, and the corporation. *Scientific American*, 265(3):128–136.

McCabe, F. G. (1992). *Logic and Objects.* Prentice-Hall, Englewood Cliffs, NJ.

McCarthy, J. (1987). Generality in artificial intelligence. *Communications of the ACM*, 30(12):1030–1035.

McCarthy, J. and Hayes, P. J. (1987). Some philosophical problems from the standpoint of AI. In Ginsberg, M., editor, *Readings in Nonmonotonic Reasoning.* Morgan Kaufmann.

Moon, D. A. (1989). The COMMON LISP object-oriented progamming lanuage. In Kim, W. and Lochovsky, F. H., editors, *Object-oriented concepts, Databases, and Applications*, pages 49–78. ACM Press.

Motro, A. (1987). Superviews: virtual integration of multiple databases. *IEEE Software Engineering*, 13(7):785–798. view integration.

Motro, A. and Yuan, Q. (1990). Querying database knowledge. In *Proceedings of the ACM SIGMOD Conference*, pages 173–183.

Mumick, I. S. and Pirahesh, H. (1994). Implementation of magic-sets in a relational database system. In *Proc. ACM SIGMOD*, pages 103–114, Minneapolis, MN.

Naiman, C. F. and Ouskel, A. M. (1995). A classification of semantic conflicts in heterogeneous database systems. *J. of Organizational Computing*, 5(2):167–193.

Papakonstantinou, Y., Garcia-Molina, H., and Widom, J. (1995). Object exchange across heterogeneous information sources. In *Proc IEEE International Conference on Data Engineering*.

Peckham, J. and Maryanski, F. (1988). Semantic data models. *ACM Computing Surveys*, 20(3):153–189.

Przymusinski, T. C. (1987). On the declarative semantics of deductive databases and logic programs. In Minker, J., editor, *Foundations of Deductive Databases and Logic Programming*, chapter 5, pages 193–216. Morgan-Kaufmann Publishers.

Qu, J. F. (1996). Data wrapping on the world wide web. Technical Report CISL WP#96-05, Sloan School of Management, Massachusetts Institute of Technology.

Quass, D., Rajaraman, A., Sagiv, Y., ullman, J., and Widom, J. (1995). Querying semistructured heterogeneous information. In *Proc International Conference on Deductive and Object-Oriented Databases*.

Reiter, R. (1978). On closed world data bases. In Gallaire, H. and Minker, J., editors, *Logic and Data Bases*, pages 55–76. Plenum Press.

Scheuermann, P., Yu, C., Elmagarmid, A., Garcia-Molina, H., Manola, F., McLeod, D., Rosenthal, A., and Templeton, M. (1990). Report on the workshop on het-

erogeneous database systems. *ACM SIGMOD RECORD*, 19(4):23–31. Held at Northwestern University, Evanston, Illinois, Dec 11–13, 1989. Sponsored by NSF.

Sciore, E., Siegel, M., and Rosenthal, A. (1992). Context interchange using meta-attributes. In *Proceedings of the 1st International Conference on Information and Knowledge Management*, pages 377–386.

Sciore, E., Siegel, M., and Rosenthal, A. (1994). Using semantic values to facilitate interoperability among heterogeneous information systems. *ACM Transactions on Database Systems*, 19(2):254–290.

Seshadri, P., Hellerstein, J. M., Pirahesh, H., Leung, T. C., Ramakrishnan, R., Srivastava, D., Stuckey, P. J., and Sudarshan, S. (1996). Cost-based optimization for magic: algebra and implementation. In *Proc. ACM SIGMOD*, pages 435–446, Montreal, Quebec, Canada.

Shanahan, M. (1989). Prediction is deduction but explanation is abduction. In *Proc. of the IJCAI-89*, pages 1055–1060.

Sheth, A. and Kashyap, V. (1992). So far (schematically) yet so near (semantically). In Hsiao, D. K., Neuhold, E. J., and Sacks-Davis, R., editors, *Proceedings of the IFIP WG2.6 Database Semantics Conference on Interoperable Database Systems (DS-5)*, pages 283–312, Lorne, Victoria, Australis. North-Holland.

Sheth, A. P. and Larson, J. A. (1990). Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236.

Siegel, M. and Madnick, S. (1991). A metadata approach to solving semantic conflicts. In *Proceedings of the 17th International Conference on Very Large Data Bases*, pages 133–145.

Smith, J. and Smith, D. (1977). Database abstractions: aggregation and generalization. *ACM Transactions on Database Systems*, 2(2):105–133.

Sterling, L. and Shapiro, E. (1994). *The Art of Prolog.* MIT Press, Cambridge, MA, 2nd edition.

Stroutstrup, B. (1986). *The C++ Programming Language.* Addison-Wesley, Readings, MA.

Templeton, M., Brill, D., Dao, S. K., Lund, E., Ward, P., Chen, A. L. P., and MacGregor, R. (1987). Mermaid — a front end to distributed heterogeneous databases. *Proceedings of the IEEE*, 75(5):695–708.

Tomasic, A., Raschid, L., and Valduriez, P. (1995). Scaling heterogeneous databases and the design of DISCO. In *Proceedings of the 16th International Conference on Distributed Computing Systems*, Hong Kong.

Toni, F. (1995). *Abductive Logic Programming.* PhD thesis, Department of Computing, Imperial College, University of London.

Ullman, J. (1985). Implementation of logical query languages for databases. *ACM Transactions on Database Systems*, 10(3):289–321.

Ullman, J. (1991a). *Principles of database and knowledge-base systems, Vol I.* Computer Science Press.

Ullman, J. (1991b). *Principles of database and knowledge-base systems, Vol II.* Computer Science Press.

Van Gelder, A., Ross, A., and Schlipf, J. S. (1988). The well-founded semantics for general logic program. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 221–230.

Ventrone, V. and Heiler, S. (1991). Semantic heterogeneity as a result of domain evolution. *ACM SIGMOD Record*, 20(4):16–20.

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

Wiederhold, G. (1993). Intelligent integration of information. In *Proceedings of the ACM SIGMOD Conference*, pages 434–437.

Wolski, A. (1989). LINDA: A system for loosely integrated databases. In *Proceedings of the Fifth International Conference on Data Engineering*, pages 66–73.

Zdonik, S. B. and Maier, D., editors (1990). *Readings in Object-oriented Database Systems*. Morgan Kaufmann, San Mateo, CA.