Context Interchange: Representing and Reasoning about Data Semantics in Heterogeneous Systems

Cheng Hian Goh, Stephane Bressan, Stuart E. Madnick, Michael D. Siegel

Sloan WP# 3928    CISL WP# 96-06
December 1996

The Sloan School of Management
Massachusetts Institute of Technology
Cambridge, MA 02142
Context Interchange: Representing and Reasoning about Data Semantics in Heterogeneous Systems*

Cheng Hian Goh†  Stéphane Bressan  Stuart E. Madnick  Michael D. Siegel
Sloan School of Management
Massachusetts Institute of Technology
Email: {chgoh,sbressan,smadnick,msiegel}@mit.edu

December 5, 1996

Abstract

The Context Interchange (COIN) strategy [44; 48] 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.

Keywords: Context, heterogeneous databases, logic and databases, mediators, semantic interoperability.

*This work is supported in part by ARPA and USAF/Rome Laboratory under contract F30602-93-C-0160, the International Financial Services Research Center (IFSRC), and the PROFIT project at MIT.
†Financial support from the National University of Singapore is gratefully acknowledged.
1 Introduction

The last few years have witnessed an unprecedented growth in the number of information sources (traditional database systems, data feeds, web-sites, and applications providing structured or semi-structured data on requests) and receivers (human users, data warehouses, and applications that make data requests). This is spurred on by a number of factors: for example, ease of access to the Internet, which is emerging as the de facto global information infrastructure, and the rise of new organizational forms (e.g., adhocracies and networked organizations [36]), which mandated new ways of sharing and managing information. The advances in networking and telecommunications technologies have led to increased physical connectivity (the ability to exchange bits and bytes), but not necessarily logical connectivity (the ability to exchange information meaningfully). This problem is traditionally referred to as the need for semantic interoperability [47] among autonomous and heterogeneous systems.

1.1 Context Interchange: Background

The goal of this paper is to describe 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 [48; 44]. 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 exist 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 [33], VIP-MDBMS [29]), 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 [30], Mermaid [49]).

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 [44], 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 [48]). 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.

1.2 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\(^1\) of the Context Interchange approach described in [48; 44] 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\(^2\) framework, based on the COIN data model, which is a customized subset of the deductive object-oriented model called Gulog\(^2\) [15]. 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 among data in different contexts. In particular, a COIN framework can be translated to a normal program [34] (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

\(^1\)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.

\(^2\)Gulog is itself a variant of F-logic [28].
be processed in an identical manner. As an alternative to the classic deductive framework, we investigate the adoption of an abductive framework [27] for query processing. Interestingly, although abduction and deduction are “mirror-images” of each other [14], 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 [7].

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 [28] 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 [37], 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 [21]. 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. 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 medi-
atated 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 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 infrastruc-
ture 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 com-
plex contexts can be derived through a hierarchical composition operator [5]. 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).

1.3 Organization of this Paper

The rest of this paper is organized as follows. Following this introduction, we present a motiva-
tional example which is used to highlight selected features of the Context Interchange strategy.
We demonstrate in Section 3 the novelty of our approach by comparing it with a variety of
other classical and contemporary approaches. Section 4 describes 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. Section 5 introduces the abduction framework and
illustrates how abductive inference is used as the basis for obtaining intensional answers, and
in particular, mediated answers corresponding to naive queries (i.e., those formulated while

---

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

4 This prototype is accessible from any WWW-client (e.g., Netscape Browser) and can be demonstrated upon
request.
assuming that sources are homogeneous). Section 6 describes a *meta-logical* extension to the COIN framework which allows changes to be incorporated in a graceful manner. Section 7 describes the prototype system which provides integrated access to a variety of heterogeneous sources while adopting the framework which we have described. We conclude in Section 8 with a number of suggestions for further work.

2 Motivational Example

Consider the scenario shown in Figure 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\(^5\). 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\(^6\):

\[
Q1: \quad \text{SELECT } r1.\text{cname}, r1.\text{revenue} \text{ FROM } r1, r2 \\
\text{WHERE } r1.\text{cname} = r2.\text{cname} \text{ AND } r1.\text{revenue} > r2.\text{expenses};
\]

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 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,

\(^5\) Throughout this paper, we make the assumption that the relational data model is adopted to be the *canonical data model* \([47]\): 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.

\(^6\) 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).
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.

In the rest of this section, we present an overview of the "functional" features of the Context Interchange strategy in providing integrated access to data in heterogeneous environments such as illustrated above. Our discussion 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. Bear in mind that these discussions are aimed at providing the underlying intuition without
necessarily being precise at all times; formal definitions of the underlying representational and inference formalisms are postponed to Sections 4, 5, and 6 of this paper.

2.1 Functional Features of Context Interchange: A User Perspective

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 [54], 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:

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

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', 9600000>.

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

\[
\text{CREATE VIEW v1 (cname, profit) AS}
\]
\[
\text{SELECT r1.cname, r1.revenue - r2.expenses}
\]
\[
\text{FROM r1, r2}
\]
\[
\text{WHERE r1.cname = r2.cname;}
\]

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

\[
\text{VQ1: SELECT cname, profit FROM v1}
\]
\[
\text{WHERE profit > 0;}
\]

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:\footnote{Sciore et al. \cite{43} 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}
Q2: \[
\text{SELECT } r1.\text{cname}, r1.\text{revenue}.\text{scaleFactor} \text{ IN } c1,
\text{ IN } c2 \text{ FROM } r1
\]
\[
\text{WHERE } r1.\text{revenue}.\text{scaleFactor} \text{ IN } c1 <> r1.\text{revenue}.\text{scaleFactor} \text{ IN } c2;
\]

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 \textit{knowledge-level queries}, as opposed to \textit{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: \[
\text{SELECT } r1.\text{cname}, 1000, 1 \text{ FROM } r1
\]
\[
\text{WHERE } r1.\text{currency} = 'JPY';
\]

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 1, we would obtain the singleton answer \('<\text{NTT}', 1000, 1>'.

\section*{Extensional versus Intensional Answers}

Yet another feature of Context Interchange is that \textit{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 \textit{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 [24]. 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 [40; 45]. 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 [7]. 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.

2.2 Functional Features of Context Interchange: A System Perspective

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\(^8\)), 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 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.

\(^8\)For an account of why this seemingly strange phenomenon may be more common than is widely believed to be, see [52].
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.

![Domain Model Diagram]

Figure 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.

Figure 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 [8] 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\(^9\)

\[
X':\text{companyFinancials}, \text{currency}(c_2, X'):\text{currencyType} \leftarrow \\
\text{currency}(c_2, X')[\text{value}(c_2) \rightarrow \text{'USD'}].
\]

The method $\text{currency}$ is said to be a modifier because it modifies how the value of a semantic-object is reported. A detailed presentation of the details of this framework will be presented later in Section 4.

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 modifiers 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,

---

\(^9\)The syntax of this language is defined in Section 4 and corresponds to that of Gulog [15].
all instances of \texttt{companyFinancials} would inherit the same scale-factor since every instance of \texttt{companyFinancials} is an instance of \texttt{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.

\section*{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 \texttt{moneyAmt} from one scale-factor ($F$ in context $C$) to another ($F_1$ in context $c_1$) can be defined as follows:

\[
X' : \texttt{moneyAmt} \vdash \\
X'[\text{cvt}(c_1)@\text{scaleFactor}, C, U\rightarrow V] \leftarrow X'[\text{scaleFactor}(c_1)\rightarrow [\text{value}(c_1)\rightarrow F_1]], \\
X'[\text{scaleFactor}(C)\rightarrow [\text{value}(c_1)\rightarrow F]], V = U * F/F_1.
\]

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 \texttt{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 \texttt{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.

\section*{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 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 currency 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:

\[
\text{currencyType}[\text{format(ctx)} \Rightarrow \text{semanticString}].
\]

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'):

\[
X: \text{currencyType}, \text{format}(c_0, X): \text{semanticString} \vdash \\
\text{format}(c_0, X)[\value(c_0) \rightarrow \text{'abbreviated'}].
\]

\[
X: \text{currencyType} \vdash X[\text{cvt}(c_0)@C, U \rightarrow V] \leftarrow \ldots (\text{body})
\]

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

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

which distinguishes it as having a different format.

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

In the preceding section, 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\textsuperscript{10}. 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 \textit{theory of contexts} [37] (as in Carnot [11], and more recently, [18]);

- \textit{encapsulation} [3] of semantic knowledge in a hierarchy of \textit{rich data types} which are refined via \textit{sub-typing} (as in several object-oriented multidatabase systems, the archetype of which is Pegasus [2]);

- adoption of a \textit{deductive or object-oriented formalism} [28; 15] (as in the ECRC Multidatabase System [26] and DISCO [50]);

- provision of value-added services through the use of \textit{mediators} [54] (as in TSIMMIS [20]);

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

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 [32]) 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 [2], the ECRC Multidatabase Project [26], and DISCO [50] 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

\textsuperscript{10}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.
optimized using traditional query optimizers or be executed by the query subsystem of classical (relational) query subsystems without changes.

In the Carnot system [11], 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 [31]. A similar approach is adopted in [18], where it is suggested that domain-specific *ontologies* [22], 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* [23] 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 [41; 42] 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.

4 The COIN Data Model: Structure, Language, and Framework

In [37], 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:
which suggests that the statement $\sigma$ is true ("ist") in the context $c$, this statement itself being asserted in an outer context $\tilde{c}$. Lifting axioms$^{12}$ are used to describe the relationship between statements in different contexts. These statements are of the form

$$\tilde{c}: \quad \text{ist}(c, \sigma) \leftrightarrow \text{ist}(c', \sigma')$$

which suggests that "$\sigma$ in $c$ states the same thing as $\sigma'$ 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 1, we may now write:

$$\tilde{c}: \quad \text{ist}(c_1, r_1('NTT', 1000000, 'JPY')).$$
$$\tilde{c}: \quad \text{ist}(c_2, r_2('NTT', 5000000)).$$

The context $\tilde{c}$ 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 [48; 44], 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 $r_1$ (as shown in Figure 1) simultaneously describes the revenue of IBM and NTT while being reported in different currencies and scale-factors. Thus, the statements

\[ ^{11}\text{In the words of Guha [23], contexts represents "the reification of the context dependencies of the sentences associated with the context." They are said to be "rich-objects" in that "they cannot be defined or completely described" [38]. Consider, for instance, the context associated with the statement: "There are four billion people living on Earth". To fully qualify the sentence, we might add that it assumes that the time is 1991. However, this certainly is not the only relevant assumption in the underlying context, since there are implicit assumptions about who is considered a "live person" (are fetuses in the womb alive?), or what it means to be "on earth" (does it include people who are in orbit around the earth?)\]

\[ ^{12}\text{also called articulation axioms in Cyc/Carnot [11].}\]
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. This will be described in further details in the next section.

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 Section 2.

4.1 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 [3], 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 complex 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 sk0 is the oid for the object representing the revenue of NTT, it is perfectly legitimate for both

(1) \text{ist}(c_1, \text{value}(sk_0,1000000)) \text{; and}

(2) \text{ist}(c_2, \text{value}(sk_0,9600000))

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\(^\text{13}\). 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.

\begin{definition}
Let \(t\) be an oid-term corresponding to a semantic-object of the semantic-type \(\tau\), and suppose the value of \(t\) is given in context \(c_s\). For any context represented by \(C\), we have

\[
\text{ist}(C, \text{value}(t, X) \leftarrow f_{cv}(t, c_s, X') = X) \leftrightarrow \text{ist}(c_s, \text{value}(t, X')).
\]

We refer to \(f_{cv}\) as the \textit{conversion function} for \(\tau\) in context \(C\), and say that \(X\) is the value of \(t\) in context \(C\), and that it is derived from context \(c_s\).

\end{definition}

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 \textit{composite conversion function}, and the latter as \textit{atomic conversion functions}. Suppose

\(^\text{13}\)A predicate-calculus language is used in the discussion here since it provides better intuition for most readers. The \textit{COIN} language, for which properties are modeled as “methods” (allowing us to write \texttt{sk0/value}→1000000 as opposed to \texttt{value(sk0,1000000)}), will be formally defined in Section 4.2.
modifiers of a semantic-type $\tau$ are $m_1, \ldots, m_k$, and $f_{cvt}$ is a composite conversion function for $\tau$. It follows that if $t$ is an object of type $\tau$, then

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

where $f_{cvt}^{(j)}$ 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

$$(t \oplus t') \Leftrightarrow (\text{value}(t, X) \land \text{value}(t', X') \land X \oplus X')$$

In particular, we say that $t$ and $t'$ are semantically-equivalent in context $c$ if $ist(c, t \equiv t')$.

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, $ist(C, \text{value}(1000\,000, 1000\,000))$ is true irregardless of what $C$ may be. Suppose we know that $ist(c_1, \text{value}(sk_0, 1000\,000))$, where $sk_0$ refers to the revenue of NTT as before. The expression

$$sk_0 \preceq 5\,000\,000$$

will therefore evaluate to "true" in context $c_1$ but not context $c_2$, since $ist(c_2, \text{value}(sk_0, 9\,600\,000))$. 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 $\text{companyFinancials}$ (see Section 5.3).

### 4.2 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* [15][14]. Rather than making inferences using a context logic (see, for

---

[14] Gulog differs from F-logic [28] 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.
example, [6]), we introduce “context” as first-class objects and capture variations in different contexts through the use of parameterized methods. For example, the context-formula \( \text{ist}(c_1, \text{value}(sk_0, 1\,000\,000)) \) can be equivalently written as \( sk_0[\text{value}(c_1) \rightarrow 1\,000\,000] \) where \( \text{value}(c_1) \) represents a (single-valued) method. This simplification is possible because of our 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 [34], 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 \( T_S \) and \( T_P \) 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., \( \text{value} \) and \( \text{cvt} \)); (5) an infinite set of variables; (6) the usual logical connectives and quantifiers \( \land, \lor, \forall, \exists, \neg \), 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 \( \tau \) and \( \tau' \) are type symbols, then \( \tau :: \tau' \) is a type declaration. We say that \( \tau \) is a subtype of \( \tau' \), and conversely, that \( \tau' \) is a supertype of \( \tau \). For any type symbol \( \tau'' \) such that \( \tau' :: \tau'' \), \( \tau \) is also a subtyping of \( \tau'' \).

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

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

• if \( \tau \) is a semantic-type, and \( \tau_1, \tau_2 \) are primitive types, then \( \tau[cvt(\text{ctx}@\text{ctx}, \tau_1 \Rightarrow \tau_2)] \) is a \textit{compound conversion declaration}. We say that the signature of the compound conversion for \( \tau \) is \( \tau \times \tau_1 \Rightarrow \tau_2 \).

• if \( \tau \) is a semantic-type, \( m \) is a modifier defined on \( \tau \), and \( \tau_1, \tau_2 \) are primitive types, then \( \tau[cvt(\text{ctx}@m, \text{ctx}, \tau_1 \Rightarrow \tau_2)] \) is a \textit{atomic conversion declaration}. We say that the signature of the atomic conversion of \( m \) for \( \tau \), is \( \tau \times \tau_1 \Rightarrow \tau_2 \).

• if \( \tau \) is a semantic-type, \( \tau_1 \) is a primitive-type and \( c \) is a context symbol, then \( \tau[value(\text{ctx}) \Rightarrow \tau_1] \) is a \textit{value declaration}. We say that the signature of the value for \( \tau \) is given by \( \tau \Rightarrow \tau_1 \).

Declarations for attributes, modifiers, conversions, and the built-in method value are collectively referred to as \textit{method declarations}.

\textbf{Definition 4} An \textit{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 \textit{predicate atom}.

• if \( m \) is an attribute symbol with signature \( \tau \rightarrow \tau' \) and \( t, t' \) are of (inherited) types \( \tau, \tau' \) respectively, then \( t[m \rightarrow t'] \) is an \textit{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 \( \tau, \tau' \) respectively, then \( t[m(c) \rightarrow t'] \) is a \textit{modifier atom}.

• if the compound conversion function for \( \tau \) has signature \( \tau \times \tau_1 \Rightarrow \tau_2 \), \( t, t_1, t_2 \) are of (inherited) types \( \tau, \tau_1, \tau_2 \) 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 \textit{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 \( \tau, \tau_1, \tau_2 \) respectively, and \( t_c \) is a context term, then \( t[cvt(c)@m, t_c, t_1 \Rightarrow t_2] \) is a \textit{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 \( \tau, \tau_1 \), then \( t[value(c) \rightarrow t'] \) is a \textit{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

- \( t[m_1 \rightarrow t_1; \ldots; m_k \rightarrow t_k] \) as a shorthand for the conjunct \( t[m_1 \rightarrow t_1] \land \cdots \land t[m_k \rightarrow t_k] \);
- \( t[m \rightarrow t_1]\) as a shorthand for \( t[m \rightarrow t_1] \land t_1 \); and
- \( t : \tau[m \rightarrow \tau'] \) as a shorthand for \( t : \tau \land t[m \rightarrow \tau'] \).

Well formed formulas can be defined inductively in the same manner as in first-order languages [34]; specifically,

- an atom is a formula;
- if \( \phi \) and \( \varphi \) are formulas, then \( \neg \phi, \phi \land \varphi \) and \( \phi \lor \varphi \) are all formulas;
- if \( \phi \) is a formula and \( X \) is a variable occurring in \( \phi \), 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

\[ \Gamma \vdash A \leftarrow B_1, \ldots, B_n \]

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 \( \Gamma \) must contain the object declarations for all oid-terms in the head. Otherwise, \( \Gamma \) may be omitted altogether.

### 4.3 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 \( <S, \mu, \mathcal{E}, D, C> \) where
• $S$, 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 $s_i$.

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

• $D$, the domain model, is a set consisting of declarations. Intuitively, declarations in the domain model identify the types, methods, and predicates which are known.

• $E$, 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 $S$. $E_i$ 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 $r_{ij}$.

• $C$, 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 1 can be represented in a COIN framework $F = <S,\mu,E,D,C>$. Figures 3 and 4 present a partial codification which we will elaborate briefly below:

• The contents of the source set $S$ 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 $cname \rightarrow \{\text{revenue, currency}\}$ on the attributes in $r_1$.

• The function $\mu$ is defined as a relation on $S \times C$: thus, source $s_1$ is mapped to context $c_1$, whereas $s_2$ and $s_3$ are both mapped to context $c_2$.
Source set \( S \)

\[
\begin{align*}
    s_1 & := \{ r_1('IBM', 1\,000\,000, 'USD'), r_1('NTT', 1\,000\,000, 'JPY') \}. \\
    R_1 & = R_2 \leftarrow r_1(N, R_1), r_1(N, R_2), \}
    Y_1 & = Y_2 \leftarrow r_1(N, Y_1), r_1(N, Y_2), \}
    s_2 & := \{ r_2('IBM', 1\,500\,000), r_2('NTT', 5\,000\,000) \}. \\
    E_1 & = E_2 \leftarrow r_2(N, E_1), r_2(N, E_2), \}
    s_3 & := \{ r_3('USD', 'JPY', 104.0), r_3('JPY', 'USD', 0.0096) \}. \\
    T_1 & = T_2 \leftarrow r_3(X, Y, T_1), r_3(X, Y, T_2) . \}
\end{align*}
\]

Source-to-Context Mapping \( \mu \)

\[
\{ \mu(s_1, c_1), \mu(s_2, c_2), \mu(s_3, c_3) \}\]

Domain model \( D \)

```
/* type declarations */
semanticNumber :: T_S.
semanticString :: T_S.
moneyAmt :: semanticNumber.
companyFinancials :: moneyAmt.
currencyType :: semanticString.
companyName :: semanticString.

/* attribute declaration */
companyFinancials[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,(currencyType,number) => number].
moneyAmt[cvt(ctx)@ctx,currencyType,number => number].

/* predicate declarations */
r_1(companyName,companyFinancials,currencyType).  \( r_1 \text{ (varchar,integer,varchar)} \).
r_2(companyName,companyFinancials).  \( r_2 \text{ (varchar,integer)} \).
r_3(currencyType,currencyType,semanticNumber).  \( r_3 \text{ (varchar,varchar,real)} \).
```
• The domain model $\mathcal{D}$ consists of two parts. The left-half (as seen in Figure 3) 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.

• 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., $f_{r2#expenses}$) 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., $f_{r2#expenses}('NTT')$) would in fact identify a unique "cell" in the relation as shown in Figure 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 $f_{r1#revenue}(-)$ 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 $\mu$; 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 $C$ is given by \{c_1 := C_1, c_2 := C_2\} and is defined by the axioms shown in Figure 5. 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:

$$\forall X' : companyFinancials \exists F' : number \vdash$$

$$(X'[scaleFactor(c_1) \rightarrow F']) \land$$

$$(F'[value(c_1) \rightarrow 1000] \leftarrow X'[currency(c_1) \rightarrow Y'] \land Y' \equiv \text{JPY'}) \land$$
Elevation Axioms $E_1$ of $E$

\[
\begin{align*}
    r_1'(fr1#cname(X_1), fr1#revenue(X_1), fr1#currency(X_1)) &\leftarrow r_1(X_1, -). \\
    fr1#cname(_) &\text{: companyName.} \\
    fr1#revenue(_) &\text{: companyFinancials.} \\
    fr1#revenue(X_1)[company \rightarrow fr1#cname(X_1)] &
    \begin{array}{l}
        \text{fr1#currency(_)} &\text{: currencyType.} \\
        fr1#cname(X_1)[value(C)\rightarrow X_1] &\leftarrow r_1(X_1, -), \mu(s_1, C). \\
        fr1#revenue(X_1)[value(C)\rightarrow X_2] &\leftarrow r_1(X_1, X_2, -), \mu(s_1, C). \\
        fr1#currency(X_1)[value(C)\rightarrow X_3] &\leftarrow r_1(X_1, X_3, -), \mu(s_1, C).
    \end{array}
\end{align*}
\]

Elevation Axioms $E_2$ of $E$

\[
\begin{align*}
    r_2'(fr2#cname(X_1), fr2#expenses(X_1)) &\leftarrow r_2(X_1, -). \\
    fr2#cname(_) &\text{: companyName.} \\
    fr2#expenses(_) &\text{: companyFinancials.} \\
    fr2#expenses(X_1)[company \rightarrow fr2#cname(X_1)] &
    \begin{array}{l}
        fr2#expenses(X_1)[value(C)\rightarrow X_1] &\leftarrow r_2(X_1, -), \mu(s_2, C). \\
        fr2#expenses(X_1)[value(C)\rightarrow X_2] &\leftarrow r_2(X_1, X_2, -), \mu(s_2, C).
    \end{array}
\end{align*}
\]

Elevation Axioms $E_3$ of $E$

\[
\begin{align*}
    r_3'(fr3#fromCur(X_1, X_2), fr3#toCur(X_1, X_2), fr3#exchangeRate(X_1, X_2)) &\leftarrow r_3(X_1, X_2, -). \\
    fr3#fromCur(\text{-}, \text{-}) &\text{: currencyType.} \\
    fr3#toCur(\text{-}, \text{-}) &\text{: currencyType.} \\
    fr3#exchangeRate(\text{-}, \text{-}) &\text{: semanticNumber.} \\
    fr3#fromCur(X_1, X_2)[value(C)\rightarrow X_1] &\leftarrow r_3(X_1, X_2, -), \mu(s_3, C). \\
    fr3#toCur(X_1, X_2)[value(C)\rightarrow X_2] &\leftarrow r_3(X_1, X_2, -), \mu(s_3, C). \\
    fr3#exchangeRate(X_1, X_2)[value(C)\rightarrow X_3] &\leftarrow r_3(X_1, X_2, X_3, \mu(s_3, C).
\end{align*}
\]

Figure 4: Elevation set corresponding to the motivational example
Context c1:

/* modifier value assignments */
X' : companyFinancials ⊨ X'[scaleFactor(c1) → scaleFactor(c1, X')].
X' : companyFinancials, scaleFactor(c1, X') : number ⊨
    scaleFactor(c1, X')[value(c1) → 1] ← X'[currency(c1) → Y'], Y' ∈ JPY'.
X' : companyFinancials, scaleFactor(c1, X') : number ⊨
    scaleFactor(c1, X')[value(c1) → 1000] ← X'[currency(c1) → Y'], Y' ∈ JPY'.
X' : companyFinancials ⊨ X'[currency(c1) → currency(c1, X')].
X' : companyFinancials, currency(c1, X') : currencyType ⊨
    currency(c1, X')[value(c1) → Y] ← X'[company → N', r(N1, R', Y'), N0 ≡ N1,
    Y'[value(c1) → Y].

/* conversion function definitions */
X' : moneyAmt ⊨
    X'[cvt(c1)@C, U→V] ← X'[cvt(c1)@scaleFactor, C, U→W],
    X'[cvt(c1)@currency, C, W→V].
X' : moneyAmt ⊨
    X'[cvt(c1)@scaleFactor, C, U→V] ← X'[scaleFactor(c1) → [value(c1) → F]],
    X'[scaleFactor(C) → [value(c1) → F]], V = U * F / F1.
X' : moneyAmt ⊨
    X'[cvt(c1)@currency, C, U→V] ← X'[currency(c1) → Y'], X'[currency(C) → Y'],
    Y' ∈ JPY', V = U.
X' : moneyAmt ⊨
    X'[cvt(c1)@currency, C, U→V] ← X'[currency(c1) → Y'], X'[currency(C) → Y'],
    Y' ∈ JPY', r(Y', Y)', Y' ≡ Y', Y' ≡ Y',
    V = U * R.

Context c2:

/* modifier value assignments */
X' : companyFinancials ⊨ X'[scaleFactor(c2) → scaleFactor(c2, X')].
X' : moneyAmt, scaleFactor(c2, X') : number ⊨ scaleFactor(c2, X')[value(c2) → 1].
X' : companyFinancials ⊨ X'[currency(c2) → currency(c2, X')].
X' : moneyAmt, currency(c2, X') : currencyType ⊨
    currency(c2, X')[value(c2) → 'USD'].

/* conversion definitions are similar to c1 and omitted for brevity */

Figure 5: Context sets for C for the motivational example at hand.
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:

\[
X' : companyFinancials \vdash \\
X'[scaleFactor(c_1) \rightarrow f_{scaleFactor(c_1)}(X')].
\]

\[
X' : companyFinancials, f_{scaleFactor(c_1)}(X') : number \vdash \\
f_{scaleFactor(c_1)}(X')[value(c_1) \rightarrow 1000] \leftarrow X'[currency(c_1) \rightarrow Y'], Y' \overset{c_2}{=} \text{JPY}'.
\]

\[
X' : companyFinancials, f_{scaleFactor(c_1)}(X') : number \vdash \\
f_{scaleFactor(c_1)}(X')[value(c_1) \rightarrow 1] \leftarrow X'[currency(c_1) \rightarrow Y'], Y' \overset{c_2}{=} \text{JPY}'.
\]

where \( f_{scaleFactor(c_1)} \) is a unique Skolem function; for notational simplicity, we replace \( f_{scaleFactor(c_2)}(X') \) with the term \( scaleFactor(c_2, X') \). Currency values corresponding to instances of \( companyFinancials \) are obtained directly from the extensional relation \( r_1 \) as shown in Figure 5. In this instance, it is necessary to reference an extensional relation because "metadata" 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 5, 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 \textbf{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\(^\text{15}\).

5 Query Answering as Abductive Inferences

Following the same algorithm outlined in [1], any collection of COIN clauses can be translated to *Datalog with negation* (*Datalog\(^{neg}\)*) (or equivalently, *normal Horn program* [34]), for which the semantics as well as computation procedures have been widely studied [51]\(^\text{16}\). In this section, we explore an alternative approach based on *abductive reasoning*. The *abductive framework* provides us with *intensional* (as opposed to *extensional*) answers to a query\(^\text{17}\). We describe this abductive framework below and the relationship between query mediation in a COIN framework and query answering in an abductive framework. In the interest of space, we assume some familiarity with logic programming at the level of [34] in the ensuing discussion, and for most part, shall remain faithful to the notations therein.

5.1 The Abductive Framework

*Abduction* 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 \( A \).

*Abductive logic programming* (ALP) [27] is an extension of *logic programming* [34] to support abductive reasoning. Specifically, an *abductive framework* [17] is a triple \( \langle T, A, I \rangle \) where \( T \) is a theory, \( I \) is a set of integrity constraints, and \( A \) is a set of predicate symbols, called *abducible* predicates. Given an abductive framework \( \langle T, A, I \rangle \) and a sentence \( \exists \bar{X}q(\bar{X}) \) (the *observation*), the *abductive task* can be characterized as the problem of finding a substitution \( \theta \) and a set of abducible \( \Delta \), called the *abductive explanation* for the given observation, such that

\[
\begin{align*}
(1) \quad & T \cup \Delta \models q(\bar{X})\theta, \\
(2) \quad & T \cup \Delta \text{ satisfies } I; \text{ and}
\end{align*}
\]

\(^{15}\)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 [13].

\(^{16}\)The fact that “object-based logics” can be encoded in classical predicate logic has been known for a long time (see for example, [9]). 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.

\(^{17}\)This change in perspective is beneficial for a variety of reasons (see Section 2), and will not be repeated here.
Requirement (1) states that \( \Delta \), together with \( T \), 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 \( \Delta \) in (3), "interesting" means primarily that literals in \( \Delta \) 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 \( \Delta \) to also be minimal or non-redundant.

Semantics and proof procedures for ALP have been active research topics recently (see [27] and references therein). We describe in this section an abduction procedure based on extensions to SLD resolution, called SLD+Abduction. The underlying idea is first reported in [12], and has inspired various different extensions. The account we give here follows that in [46].

We first consider SLD resolution. Given a theory \( T \) consisting of (definite) Horn clauses and a goal clause \( \leftarrow q(\vec{X}) \), and SLD-refutation of \( \leftarrow q(\vec{X}) \) is a sequence of goal clauses \( \leftarrow G_0 (= q(\vec{X})) ; \leftarrow G_1 ; \cdots ; \leftarrow G_n \) where \( \leftarrow G_n \) is the empty clause (\( \square \)) 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 \( T \). Since there may be many clauses in \( T \) 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 (e.g., in a depth-first manner).

Suppose now that there is some \( \leftarrow G_i \), whose selected literal \( g \) will not resolve with any clause in \( T \). 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 to a refutation (i.e., one which terminates in an empty clause). Given however that we are searching for a set of unit clauses \( \Delta \), such that \( T \cup \Delta \models G \), then clearly by letting \( \Delta \) 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 \( \langle T, \Delta, I \rangle \) and the abductive query \( q(\vec{X}) \), consider the sequence given by

\[
\begin{align*}
\leftarrow & G_0, \Delta_0 \text{ where } G_0 = q(\vec{X}) \text{ and } \Delta_0 \text{ is the empty set} \\
& \vdots \\
\leftarrow & G_n, \Delta_n
\end{align*}
\]

such that \( G_{i+1}, \Delta_{i+1} \) is derived from \( G_i, \Delta_i \) as follows:

- if \( g \), the selected literal of \( \leftarrow G_i \), can be resolved with a clause in \( T \), 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 \( \langle \mathcal{T}, \mathcal{A}, \mathcal{I} \rangle \). 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 \( \Delta_n \) is said to be the residue corresponding to this refutation, and constitutes the abductive answer to \( \forall (q(\bar{X}) \theta) \), where \( \theta \) is the substitution obtained from the composition of all substitutions leading to the refutation, restricted to the variables \( \bar{X} \).

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 \( j > i \)). In [16], it is 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) [10] 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 [17]. 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.
5.2 Query Answering in the COIN Framework

Figure 6 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 2.1.

Figure 6: 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\textsuperscript{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 Section 2 can be mapped to the following clausal representations:

\[
\text{CQ1: } \neg \text{answer}(N, R),
\text{answer}(N, R) \leftarrow r_1(N, R, \_), r_2(N, E), R > E
\]

and correspondingly,

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

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 \( \text{scaleFactor} \) 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 7 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 \( X \gtrless Y \).
- make all relational “joins” explicit by replacing shared variables with explicit equality using the semantic-operator \( 
\Xi \); for example, \( r_1(X, Y), r_2(X, Z) \) would be replaced with \( r_1(X_1, Y), r_2(X_2, Z), X_1 \gtrless X_2 \).
- similarly, make relational “selections” explicit; thus, \( r_1(X, a) \) will be replaced by \( r_1(X, Y), Y \gtrless a \).
- 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 \(<CQ1, c_2>\) and \(<CQ2, c_2>\), are given by

\[
CQ1': \leftarrow \text{answer}(N, R).
\]

\[
\text{answer}(N, R) \leftarrow r'_1(N'_1, R', -), r'_2(N'_2, E'), N'_1 \overset{c_2}{=} N'_2, R' \supset E',
\]

\[
N'_1[\text{value}(c_2)\rightarrow N], R'[\text{value}(c_2)\rightarrow R].
\]

and

\[
CQ2': \leftarrow \text{answer}(N, F_1, F_2).
\]

\[
\text{answer}(N, F_1, F_2) \leftarrow r'_1(N', R', -), R'[\text{scaleFactor}(c_1)\rightarrow F'_1], R'[\text{scaleFactor}(c_2)\rightarrow F'_2],
\]

\[
F'_1 \overset{c_2}{=} F'_2, N'[\text{value}(c_2)\rightarrow N], F'_1[\text{value}(c_2)\rightarrow F_1], F'_2[\text{value}(c_2)\rightarrow F_2].
\]

respectively.

**Transformations to an Abductive Framework**

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

**Definition 8** Given the COIN framework \(\mathcal{F}_C = <S, \mu, \mathcal{E}, \mathcal{D}, C>\), 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\(^{neg}\) 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 \(S\), augmented with Clark's Free Equality Axioms [10]; and

- \(\mathcal{A}\) consists of the extensional predicates defined in \(S\), the built-in predicates corresponding to arithmetic and relational (comparison) operators, and the system predicate which provides the interface for system calls.

Suppose \(\leftarrow q(\overline{X})\) 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 \(\leftarrow q(\overline{X})\) is identical in both \(\mathcal{F}_C\) and \(\mathcal{F}_A\). This is because Datalog\(^{neg}\) is a sublanguage of COIN, and any COIN query \(\leftarrow Q(\overline{X})\) can always be transformed to a Datalog\(^{neg}\) query \(\leftarrow q(\overline{X})\) by adding the Datalog\(^{neg}\)-translation of the clause \(q(\overline{X})\leftarrow Q(\overline{X})\) into the theory \(\mathcal{T}\).

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

\[
\mathcal{T} \models (q(\overline{X})\theta \leftarrow p_1, \ldots, p_m)
\]
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 $p_1 \land \cdots \land p_m$ 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 $\varphi$ 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 [39] 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 [39] 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 illustration of the preceding comments, the evaluation of $CQ2'$ in the abductive framework yields the following abductive answer:

$$\Delta = \{r_1(sk_0, sk_1, sk_2), sk_2 = 'JPY'), \theta = \{N/sk_0, F_1/1000, F_2/1\}$$

The reverse substitution $\varphi$ 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:

$$(\exists Y_0, Y_1, Y_2(r_1(Y_0, Y_1, Y_2), Y_2 = 'JPY'), \{N/Y_0, F_1/1000, F_2/1\})$$

which translates to $MQ2$ shown in Section 2. If $\{Y_0/’NTT’, Y_1/1000000, Y_2/’JPY’\}$ is an answer for the above mediated query, then the answer for the original user query is given by $\{N/’NTT’, F_1/1000, F_2/1\}$.

5.3 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 $c_1$:

$Q3: \quad$ SELECT $r1.cname, r1.revenue.scaleFactor$ IN c1$
FROM $r1$;

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

$$CQ3: \leftarrow answer(N, F).$$

$$answer(N, F) \leftarrow r1'(N', R', \_), N'[value(c_1) \rightarrow N], R'[scaleFactor(c_1) \rightarrow F'],$$
$$F'[value(c_1) \rightarrow F].$$
Figure 7 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 3, 4 and 5.

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 $\Delta$ 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, $r_1(s_{k0}, s_{k1}, s_{k2})$, is added to $\Delta$.

- At step (6), the literal $\text{scaleFactor}(c_1, f_{r1#revenue}(s_{k0}))[\text{value}(c_1) \rightarrow F]$ can be resolved with two different clauses (where $F = 1$ and $F = 1000$). One is chosen arbitrarily (in this case, $F = 1$); the other will be selected on backtracking and will eventually lead to another refutation.

- 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 5). 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, $r_1(s_{k3}, s_{k4}, s_{k5})$ to $\Delta$.

- In step (15), the equality constraint on the objects $f_{r1#cname}(s_{k0})$ and $f_{r1#cname}(s_{k3})$ leads to the constraint $s_{k0} = s_{k3}$. Since '=' is abducible (it is an evaluable predicate), it is added to $\Delta$. At this point, the functional dependency $\text{cname} \rightarrow \{\text{revenue}, \text{currency}\}$ generates further the constraints $s_{k1} = s_{k4}$ and $s_{k2} = s_{k5}$, which in turn allow us to merge the two facts $r_1(s_{k0}, s_{k1}, s_{k2})$ and $r_1(s_{k3}, s_{k4}, s_{k5})$.

- Finally, in step (17), the literal $s_{k2} = 'JPY'$ is abduced, which leads to a refutation. The abductive answer corresponding to this refutation is given by $\Delta = \{r_1(s_{k0}, s_{k1}, s_{k2}), s_{k2} = 'JPY'\}$. The substitution, restricted to variables $\{N, F\}$, is given by $\{N/s_{k0}, F/1\}$.

This intensional answer, translated to SQL, is given by:
Figure 7: One possible refutation for query CQ3. Method and functor names are abbreviated where possible (e.g., cr = currency). The resolution step labeled \( \triangleright \) is where a literal is abduced. The abductive answer corresponding to this refutation is given by \( \Delta_4 \), and the intensional answer by \( (\Delta_4, \{ N/sko, F/1 \}) \).
SELECT r1.cname, 1 FROM r1 WHERE r1.currency <> 'JPY';

On backtracking, the other solution corresponding to \( F = 1000 \) 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, 1000 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 \( \Delta \), thus resulting in an unsuccessful refutation. In this case, the mediated query MQ3' will consist of only the second select-statement in MQ3.

6 A Meta-Logical Extension to the COIN Framework

In Section 4.3, context knowledge in a COIN framework is represented by a set of separate theories (i.e., \( C = \{c_1 := C_1, \ldots, c_n := C_n\} \)). 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 \( - \), 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
is taken to mean that the statement \( \sigma \) is true in context \( c_i \). The relation \( \prec \) allows us to make incremental refinements to statements which describe what is already known 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 \( \delta_{c_i} \), such that:

\[
\text{ist}(c_i, \sigma) \leftarrow \sigma \in \delta_{c_i} \\
\text{ist}(c_i, \sigma) \leftarrow c_i \prec c_j, \text{ist}(c_j, \sigma), \text{not-overridden}(\delta_{c_i}, \sigma).
\]

The predicate \text{not-overridden} indicates that the statement \( \sigma \) 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 \( \delta_{c_i} \) is similar to that accomplished by the \text{isa} operator defined in [4].

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

\[
\text{ist}(c_j, t[m_1 \rightarrow t'] \leftarrow t[m_2 \rightarrow t']).
\]

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 \( \delta_{c_j} \) 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 \( \delta_{c_j} \). The statement \( \sigma(X) \) is said to be uninstantiated. The collection of uninstantiated axioms forms an uninstantiated context set.

**Definition 9** Let \( \delta C = \{c_0 := C_0(X), \delta_{c_1}(X), \ldots, \delta_{c_n}(X)\} \), 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 \( \{c_1, \ldots, c_{i_k}\} \) 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.,
• \( \sigma(X) \in C_{i_j}(X) \leftarrow \sigma(X) \in \delta_{c_{i_j}}(X) \)

• \( \sigma(X) \in C_{i_j}(X) \leftarrow \sigma(X) \in C_{i}(X), \text{not-overridden}(\delta_{c_{i_j}}(X), \sigma(X)) \).

The context \( c_0 \) is said to be the default context and forms the basis for the other differentials.

\[ \square \]

**Definition 10** Given \( \delta C = \{ c_0 := C_0(X), \delta c_1(X), \ldots, \delta c_n(X) \} \) and the subcontext relation \(<\). Suppose \( C_i(X) \) is the uninstantiated context set for \( c_i \) obtained inductively using Definition 9. 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 11** The extended COIN framework is a sextuple given by \( <S, \mu, E, D, \delta C, < \> \), where \( S, E, \) and \( D \) are defined as before in Definition 6, \( \delta C \) is as defined in Definition 9, and \(< \) is the subcontext relation defined on the set of contexts \( \{c_1, \ldots, c_n\} \) induced by \( \delta C \).

\[ \square \]

7 The Context Interchange Prototype

The feasibility and features of the proposed strategy have been demonstrated in a prototype which provides mediated access to both traditional structured databases and semi-structured data sources (web-sites). Our implementation leverages on the world-wide-web (WWW) in a number of ways: for providing physical connectivity across different networks and platform, in adopting a universal addressing scheme to different types of geographically-distributed resources, and for providing us with a wealth of heterogeneous data sources. As shown in Figure 8, queries submitted to the system are intercepted by a Context Mediator, which rewrites the user query to a mediated query. The Optimizer transforms this to an optimized query plan, which takes into account a variety of cost information. The optimized query plan is executed by an Executioner which dispatches subqueries to individual systems, collates the results, undertakes conversions which may be necessary when data are exchanged between two systems, and returns the answers to the receiver.
8 Conclusion

We have presented 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

\[\text{ECLiPSe} \] \text{18} \text{, which is an efficient and robust Prolog implementation distributed by the ECRC. At the heart of the Context Mediator is a meta-interpreter which implements the extended SLD+Abduction algorithm described in Section 5.1. 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\textsuperscript{agg}, 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.

\[\text{18} \text{ECLiPSe: The ECRC Constraint Logic Parallel System. More information can be obtained at http://www.ecrc.de/eclipse/}.\]
"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.

This paper is by no means the last word on Context Interchange. On the contrary, there are many interesting issues which we are only beginning to explore. We mention below two of these undertakings.

As noted in [35], 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 recently embarked on a re-implementation of the abduction procedure using Constraint Handling Rules [19], and have found much synergy between the abduction framework, semantic query optimization, and constraint logic programming [25] on the premise of similar observations which motivated the work recently presented in [53]. 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.
References


