Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Systems

by

Cheng Hian Goh

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

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Jan 1997

© Cheng Hian Goh, 1997. All rights reserved.

The author hereby grants to MIT permission to reproduce and
 distribute publicly paper and electronic copies of this thesis document
 in whole or in part, and to grant others the right to do so.

Sloan School of Management
Dec 5, 1996

Certified by.................................................................

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

Accepted by.................................................................

Birger Wernerfelt
Chairman, Departmental Committee on Graduate Students
Representing and Reasoning about Semantic Conflicts in
Heterogeneous Information Systems

by

Cheng Hian Goh

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

Abstract

The context interchange (COIN) strategy [Sciore et al., 1994, Siegel and Madnick, 1991] presents a novel perspective for mediated data access in which semantic conflicts among heterogeneous systems are not identified a priori, but are detected and reconciled by a Context Mediator through comparison of contexts associated with any two systems engaged in data exchange. In this Thesis, 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 teasing apart context knowledge from the underlying schemas whenever feasible, by enabling sources and receivers to remain loosely-coupled to one another, and by sustaining an infrastructure for data integration. The feasibility and features of this approach have been demonstrated in a prototype implementation which provides mediated access to traditional database systems (e.g., Oracle databases) as well as semi-structured data (e.g., Web-sites).

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

The adage "time flies" has never ringed more truth to me. The last four years and three months slipped by more quickly than I ever imagined possible. There were times of exhilaration and of grief, of hope and of despair, of awe and of skepticism. At this juncture when one phase of my life is giving way to another, it is time to thank the many people who have helped during my sojourn at MIT.

I am grateful to Professor Stuart Madnick for his leadership in the Context Interchange project, without which the research described in this Thesis could not have been accomplished. His endless supply of anecdotes are a source of inspiration and provided the motivation for this work. Most importantly, I thank Stu for his role as a mentor and for his faith in my abilities at the times when I am still ironing out glitches in the theory. I owe a great debt to the other members of the Thesis Committee, comprising of (in no particular order) Dr Michael Siegel, Professor Edward Sciore, and Dr Umeshwar Dayal. Michael has been incredibly generous in meeting every of my request, whether is it for office space, equipment, manpower, attendance at workshops and conferences, or even food: he will always be remembered for his charming manners and generosity. His work (with Ed) on semantic-values played an important part in the Thesis. I thank both of them for their indulgence over my (sometimes overly zealous) criticisms of the semantic-value model. I like to thank also Ed and Umesh for their time and commitment in reading drafts of my Thesis despite their busy schedule. Ed has offered many kind words which are a great source of comfort and encouragement. It was a privilege to have Umesh scrutinize the work: despite the difficulties brought about by time and geography, he has managed to soak up an incredible amount of our work and has probably left few stones unturned. All
in all, this work has benefited greatly from detailed comments from every member of the Thesis committee.

A number of other people have read various drafts of the Thesis and provided valuable feedback. These include Dr Stéphane Bressan, Professor Michael Kifer, Dr Arnie Rosenthal, Tito Pena and Tom Lee. In particular, Steph has provided numerous suggestions on how the Thesis can be improved, eventually prompting a complete rewriting incorporating many of his suggestions. He has also helped me articulate the contributions of the Thesis, often times by acting as a sounding board for me to bounce off ideas.

The Context Interchange prototype (demonstrating the feasibility of the proposed theory) is made possible through the labor of many past and present students. They include (in rough chronological order) Alan Moulton, Henry Kon, Adil Daruwala, Scott Hofmeister, Karim Hussein, Hoony Youn, Marta Jacobisiak, Jessica Qu, Tom Lee, Tito Pena, Kofi Fynn, and Andy Shum. In particular, I like to thank Tito for his implementation of the COIN-to-Datalog compiler and for the fantastic job in assembling various pieces of the initial prototype. Steph deserves a special word of thanks for teaching me the constraint propagation features in Eclipse; the enhancements he suggested has greatly improved the readability and robustness of the Context Mediator.

A multitude of people, apart from those directly involved in this research, have been a source of blessing in one way or another: I apologize for not being able to thank each one of them individually. Notwithstanding, I feel compelled to name three — Professor Wanda Orlikowski, Sharon Cayley, and Deb Small — who have gone beyond their call of duty, in showing genuine concern for my well-being at a time when I was in grief. I thank them sincerely for the kindness which they have demonstrated to me.

I like to thank the National University of Singapore for providing me with a scholarship to undergo the PhD programme at Sloan. Without their generous support, I would not have chosen this path. In particular, I am grateful to Professor Yuen Chung Kwong who was instrumental in helping me obtain the scholarship, and to Professor
Chong Chi Tat, for his continued support. I like to thank Associate Professor Ling Tok Wang, Dr Ooi Beng Chin and Dr Lu Hongjun for my induction into database research. The "logic and database" class I took many years ago with Prof Ling was probably the single-most important factor which shaped the solution approach taken in this Thesis.

I like also to express my appreciation to the extended family in Singapore, for shouldering my share of responsibility for our parents in my absence. Despite the pain they were experiencing, they have kept my father's illness from me as long as they could in order that I can remain focused on my examinations. I like to thank my wife Soh Mui and son "bear-bear", who are to me a bulwark to which I can always turn to in times of discouragement and pain. Their unconditional love and affirmation of value is the single most important thing that has helped me persevere on. Finally, I like to thank the LORD, for his goodness and mercy in sustaining each one of us every day of our lives.

I wish to dedicate this Thesis to my father (1925–1995), who had wanted to share in my prize.
## Contents

1 Introduction  
  1.1 Summary of Contributions ................................ 12  
  1.2 Thesis Outline ............................................. 15  

2 Literature Survey  
  2.1 Data Heterogeneity ........................................ 19  
  2.2 Approaches to Achieving Interoperability ............... 25  
  2.3 The Measure of a Viable Integration Strategy .......... 29  
  2.4 Discussion ............................................... 33  

3 Context Interchange By Example  
  3.1 Scenario Description ...................................... 36  
  3.2 A User Perspective of Context Interchange ............ 37  
  3.3 A System Perspective of Context Interchange .......... 43  

4 The COIN Data Model  
  4.1 Background ................................................. 52  
  4.2 Formalizing Context for Semantic Interoperability .... 58  
  4.3 The Structural Elements of COIN ......................... 60  
  4.4 The Language of COIN ..................................... 63  
  4.5 The COIN Framework ....................................... 66  
  4.6 A Meta-Logical Extension to the COIN Framework ...... 74
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Query Answering in the COIN Framework</td>
<td>78</td>
</tr>
<tr>
<td>5.1</td>
<td>Abductive Logic Programming</td>
<td>79</td>
</tr>
<tr>
<td>5.2</td>
<td>On the Relationship between Abduction and Deduction</td>
<td>81</td>
</tr>
<tr>
<td>5.3</td>
<td>The SLD+Abduction Procedure and Its Extensions</td>
<td>82</td>
</tr>
<tr>
<td>5.4</td>
<td>Query Answering in the COIN Framework</td>
<td>84</td>
</tr>
<tr>
<td>5.5</td>
<td>Illustrative Example</td>
<td>89</td>
</tr>
<tr>
<td>5.6</td>
<td>Discussion</td>
<td>92</td>
</tr>
<tr>
<td>6</td>
<td>Data Modeling in COIN</td>
<td>95</td>
</tr>
<tr>
<td>6.1</td>
<td>The Integration Scenario</td>
<td>95</td>
</tr>
<tr>
<td>6.2</td>
<td>Domain Modeling</td>
<td>98</td>
</tr>
<tr>
<td>6.3</td>
<td>Mapping Sources to the Domain Model</td>
<td>99</td>
</tr>
<tr>
<td>6.4</td>
<td>Context Definition</td>
<td>102</td>
</tr>
<tr>
<td>6.5</td>
<td>Discussion</td>
<td>109</td>
</tr>
<tr>
<td>7</td>
<td>The Context Interchange Prototype</td>
<td>112</td>
</tr>
<tr>
<td>7.1</td>
<td>The Context Interchange Prototype: Overview</td>
<td>112</td>
</tr>
<tr>
<td>7.2</td>
<td>Implementation of the Context Mediator</td>
<td>116</td>
</tr>
<tr>
<td>7.3</td>
<td>Mediated Data Access: A User's Perspective</td>
<td>121</td>
</tr>
<tr>
<td>8</td>
<td>Conclusion</td>
<td>123</td>
</tr>
<tr>
<td>8.1</td>
<td>Context Interchange vis-à-vis Traditional and Contemporary Integration Approaches</td>
<td>124</td>
</tr>
<tr>
<td>8.2</td>
<td>Future Work</td>
<td>128</td>
</tr>
</tbody>
</table>
List of Figures

2-1  A taxonomy of data conflicts proposed in this Thesis. .................. 20
2-2  Schematic conflicts resulting from the different design decisions con-
      cerning how data should be aggregated. .............................. 21
2-3  Schematic conflicts resulting from different design choices concerning
      how “generalization” is applied. ........................................ 22
2-4  A sampling of prototype systems constructed using the tightly-coupled
      (■) and the loosely-coupled (★) strategies. ............................ 26
3-1  Scenario for the motivational example. ................................. 37
3-2  The architecture of a Context Interchange system. .................... 43
3-3  An informal characterization of the Context Interchange Framework. 46
3-4  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. ................................. 48
4-1  The source set, source-to-context mapping, and domain model for the
      COIN framework corresponding to the motivational example. ........ 68
4-2  Elevation set corresponding to the motivational example ............. 71
4-3  Context sets for C for the motivational example at hand. ............ 72
5-1 A summary of how queries are processed within the Context Interchange strategy: ① transforms a (extended) SQL query to a well-formed \textsc{coin} query; ② performs the \textsc{coin} to \textsc{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.

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

6-1 Integration scenario.

6-2 A domain model for the integration scenario. (Note that \texttt{companyName}, \texttt{countryName}, \texttt{dateFormat}, etc are subtypes of the type \texttt{semanticString}. These subtype relations, along with the type \texttt{semanticString} are not shown in the diagram for clarity reasons.)

6-3 Alteration to the \textsc{disclosure} and \textsc{worldscope} schemas resulting in location-of-incorporation being reported in the \textsc{worldscope} database instead.

7-1 Architectural overview of the Context Interchange Prototype.

7-2 Context Mediator Internals.

7-3 Screen-shot of the multidatabase browser.
"There is nothing more practical than a good theory."
- Kurt Lewin

Chapter 1

Introduction

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

\[
\text{salary('Jones', 2000)}
\]

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

\(^{1}\text{context: n. 1. The part of a written or spoken statement in which a word or passage at issue occurs and that often specifies its meaning. 2. The circumstances in which a particular event occurs; situation. (The American Heritage Dictionary).}
mantic interoperability among autonomous and heterogeneous systems [Scheuermann et al., 1990, Sheth and Larson, 1990, Hurson et al., 1994].

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

- the detection and reconciliation of semantic conflicts are system services which are provided by a Context Mediator, and should be transparent to a user; and

- the provision of such a mediation service requires only that the user furnish a logical (declarative) specification of how data are interpreted in sources and receivers, and how conflicts, when detected, should be resolved, but not what conflicts exists a priori between any two systems.

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

As interesting as these ideas may be, they could remain as vague musings in the absence of a formal conceptual foundation. One attempt at identifying a conceptual basis for Context Interchange is the semantic-value model [Sciore et al., 1994], where each data element is augmented with a property-list which defines its context. This model, however, continues to be fraught with ambiguity. For example, it relied on implicit agreement on what the modifiers for different attributes are, as well as what conversion functions are applicable for different kinds of conflicts, and is silent on how different conversion definitions can be associated with distinct contexts. Defining the semantics of data through annotations attached to individual data elements tend also to be cumbersome, and there is no systematic way of promoting the sharing and
reusing of these representations. At the same time, the representational formalism remains somewhat detached from the underlying conflict detection algorithm (the \textit{subsumption algorithm} [Siegel and Madnick, 1991]). Among other problems, this algorithm requires conflict detection to be done on a pairwise basis (i.e., by comparing the context definitions for two systems at a time), and is non-committal on how a query plan for multiple sources can be constructed based on the sets of pair-wise conflicts. Furthermore, the algorithm limits meta-attributes to only a single-level (i.e., property lists cannot be nested), and are not able to take advantage of known constraints for pruning off conflicts which are guaranteed never to occur.

1.1 Summary of Contributions

We have two (intertwined) objectives in this paper. First, we aim to provide a formal foundation for the Context Interchange strategy that will not only rectify the problems described earlier, but also provide for an integration of the underlying representational and reasoning formalisms. Second, the deconstruction and subsequent reconstruction\footnote{In this \textit{deconstruction}, we tease apart different elements of the Context Interchange strategy with the goal of understanding their contributions individually and collectively. The \textit{reconstruction} examines how the same features (and more) can be accomplished differently within the formalism we have invented.} of the Context Interchange approach described in [Siegel and Madnick, 1991, Sciole et al., 1994] provides us with the opportunity to address the concern for integration strategies that are scalable, extensible and accessible.

Our formal characterization of the Context Interchange strategy takes the form of a \texttt{COIN framework}, based on the \texttt{COIN} data model, which is a customized subset of the deductive object-oriented model called Golog\footnote{Golog is itself a variant of \textit{F-logic} [Kifer et al., 1995].} [Dobbie and Topor, 1995]. \texttt{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 \texttt{COIN} provide us with a well-founded basis for making inferences about semantic disparities that exist among data in different contexts. In particular, a \texttt{COIN} framework can be translated to a \textit{normal program} [Lloyd, 1987] (equivalently, a \textit{Datalog}\textsuperscript{neg} program) for which
the semantics is well-defined, and where sound computational procedures for query answering exist. Since there is no real distinction between factual statements (i.e., data in sources) and knowledge (i.e., statements encoding data semantics) in this logical framework, both queries on data sources (data-level queries) as well as queries on data semantics (knowledge-level queries) can be processed in an identical manner. As an alternative to the classic deductive framework, we investigate the adoption of an abductive framework [Kakas et al., 1993] for query processing. Interestingly, although abduction and deduction are “mirror-images” of each other [Denecker and Schreie, 1992a], the abductive answers, computed using a simple extension to classic SLD-resolution leads to intensional answers as opposed to extensional answers that would be obtained via deduction. Intensional answers are useful in our framework for a number of conceptual and practical reasons. In particular, if the query is issued by a “naive” user under the assumption that there are no conflicts whatsoever, the intensional answer obtained can be interpreted as the corresponding mediated query in which database accesses are interleaved with data transformations required for mediating potential conflicts. Finally, by checking the consistency of the abducted answers against known integrity constraints, we show that the abducted answer can be greatly simplified, demonstrating a clear connection to what is traditionally known as semantic query optimization [Chakravarthy et al., 1990].

As much as it is a logical data model, COIN is also an “object-oriented” data model because it adopts an “object-centric” view of the world and supports many of the features (e.g., object-identity, type-hierarchy, inheritance, and overriding) commonly associated with object-orientation. The standard use of abstraction as well as structural and behavioral inheritance [Kifer et al., 1995] present many opportunities for sharing and reuse of semantic encodings. Conversion functions (for transforming the representation of data between contexts) can be modeled as methods attached to types in a natural fashion. Unlike “general purpose” object-oriented formalisms, we make some adjustments to the structure of our model by distinguishing between different kinds of objects which have particular significance for our problem domain. In particular, we distinguish between primitive-objects (which are instances of data
types native to sources and receivers) and semantic-objects (introduced in our data model to facilitate reasoning with different representations of the same data element). Disparate assumptions concerning the semantics of individual data elements are described with reference to context-objects [McCarthy, 1987]. Context-objects serve as reified representations for collections of statements about particular contexts. By providing a common reference point for anchoring descriptions of data semantics, they allow us to make inferences across multiple theories which may be mutually inconsistent.

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

The above concerns are addressed in two ways in the reconstructed Context Interchange strategy⁴. Provisions for making sources more accessible to users is accomplished by shifting the burden for conflict detection and mediation to the system; supporting multiple paradigms for data access by supporting queries formulated directly on sources as well as queries mediated by views; by making knowledge of disparate semantics accessible to users by supporting knowledge-level queries and answering with intensional answers; and by providing feedback in the form of mediated queries. Scalability and extensibility are addressed by maintaining the distinction between the representation of data semantics as is known in individual contexts, and the detection of potential conflicts that may arise when data are exchanged between two systems; by allowing contexts and schemas to remain loosely-coupled to each

⁴For the sake of brevity, further references made to the Context Interchange strategy from this point on refers to this reconstruction unless otherwise specified.
other so that data semantics can be specified with reference to complex object types in the domain model as opposed to annotations on attributes in 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 infrastructure for data integration that combines these features. As a special effort in providing such an infrastructure, we introduce a *meta-logical* extension of the COIN framework which allows sets of context axioms to be “objectified” and placed in a hierarchy, such that new and more complex contexts can be derived through a *hierarchical composition operator* [Brogi and Turini, 1991]. This mechanism, coupled with type inheritance, constitutes a powerful approach for incorporating changes (e.g., the addition of a new system, or changes to the domain model) in a graceful manner.

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

### 1.2 Thesis Outline

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

**Chapter 3** paves the way for the rest of the Thesis by exemplifying the features of the Context Interchange strategy through the use of examples. This provides an overview of *what* is being accomplished, *how* this is aided by the underlying COIN

---

\(^5\)This prototype is accessible from any WWW-client (e.g., Netscape Browser) and can be demonstrated upon request.
data model, as well as why the various features are useful. Technical discussion is deliberately kept to the minimal so that this material can be accessible to a wider audience. In so doing, we hope the benefits of the proposed integration approach can be appreciated by even the casual reader who may not want to be encumbered by the technical material.

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

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

Chapter 6 presents a case study demonstrating how the Context Interchange strategy can be used to provide mediated access in a "real world" integration scenario. This serves primarily to illustrate the types of conflicts which can be overcomed using our integration approach, while providing the context for an extended discussion on the design and specifications of appropriate domain model and context axioms.

Chapter 7 presents details on the design and implementation of the Context Interchange Prototype. The Context Mediator (the key component responsible for rewriting a user query to a mediated one) is implemented as part of a testbed which
includes a frontend for query formulation, gateways to relational databases, and wrappers for semi-structured documents served by Web-sites. A variety of data sources are currently accessible from this Prototype, allowing one to experiment with different types of queries so as to achieve a better appreciation of the features pertaining to our integration approach.

Finally, Chapter 8 concludes the Thesis with a summary of our contributions and present a comparison of our approach with both classical and contemporary integration strategies proposed in the literature. This is followed by a number of suggestions on how the current research may be extended.
Chapter 2

Literature Survey

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

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

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

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

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

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

### 2.1 Data Heterogeneity

Roughly speaking, we can distinguish between three types of data conflicts. Of these, the first two — *schematic conflicts* [Kim and Seo, 1991, Krishnamurthy et al., 1991], *semantic conflicts* [Sheth and Kashyap, 1992, Naiman and Ouskel, 1995, Garcia-Solaco et al., 1996] — have been well-documented in the literature (though there is little consensus of what each encompasses). In most instances, the distinction between the two can be characterized by differences in "structure" ("how are the data logically organized?") versus that of "interpretation" ("what do the data mean?"). This distinction however is not always crisp, since the logical organization of data often conveys semantic information. A third category of conflicts, which we shall refer to as *intensional conflicts*, are concerned with the differences over the contents of information which are present (or expected) in distinct systems. The remainder
of this section introduces a taxonomy (as illustrated in Figure 2-1) which provides a synthesis of the literature on this subject.

![Diagram of data conflicts]

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

**Schematic Heterogeneity**

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

**Data Type Conflicts**

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

Labelling Conflicts

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

<table>
<thead>
<tr>
<th>Database A</th>
<th>Database B</th>
</tr>
</thead>
<tbody>
<tr>
<td>relation Jan0196</td>
<td>relation HPP</td>
</tr>
<tr>
<td>StkCode</td>
<td>TradePrice</td>
</tr>
<tr>
<td>HPP</td>
<td>30.10</td>
</tr>
<tr>
<td>BBM</td>
<td>40.20</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>relation Jan0296</td>
<td>relation BBM</td>
</tr>
<tr>
<td>StkCode</td>
<td>TradePrice</td>
</tr>
<tr>
<td>HPP</td>
<td>30.50</td>
</tr>
<tr>
<td>BBM</td>
<td>41.00</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Database A</th>
<th>Database B</th>
</tr>
</thead>
<tbody>
<tr>
<td>relation <em>Employees</em></td>
<td>relation <em>Managers</em></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Department</td>
</tr>
<tr>
<td>Jones</td>
<td>production</td>
</tr>
<tr>
<td>Simpson</td>
<td>development</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Name</td>
<td>Department</td>
</tr>
<tr>
<td>Jones</td>
<td>production</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>relation <em>Engineers</em></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Department</td>
</tr>
<tr>
<td>Simpson</td>
<td>development</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

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

Generalization Conflicts

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

**Semantic Heterogeneity**

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

**Naming Conflicts**

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

**Scaling and Units Conflicts**

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

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

Intensional Heterogeneity

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

Domain Conflicts

Domain Conflicts refer to discrepancies in the domain, or the underlying “universe of discourse”, which is (implicitly) modeled by each component system. As an illustration, two sources may provide financial information on companies, but the first reports “all US Fortune 500 companies in the manufacturing sector”, whereas the second may report information for “all companies listed on US stock exchanges (foreign or US-incorporated) with total assets above one billion US Dollars”. In general, the extensions of two distinct components may be related in a number of ways: they may be identical, one may be a strict subset of the other, they may be disjoint, or they may overlap in some nontrivial way. Knowing what extensional data sets are provided by disparate sources is important for determining a minimal set of database requests needed to satisfy a given query: if it is known that the extension of source A is a superset of those of sources B and C, querying all three sources is clearly redundant and entails more work than necessary. Receivers too can have implicit ex-
tensions. An application may be designed around the assumption that it is operating on a collection qualified in some manner (e.g., all stocks traded on New York Stock Exchange). It is critical that queries issued by this application be understood with reference to the assumptions which bind the extensional data set in order that it can be correctly interpreted.

**Integrity Constraint Conflicts**

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

### 2.2 Approaches to Achieving Interoperability

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

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

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

- *who* is responsible for identifying what conflicts exists and how they can be circumvented; and

- *when* the conflicts are resolved.

Clearly, the two tasks are correlated since the first must precede the second. The remainder of this section describes the pertinent features of the two approaches.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Deductive (Logic-Based)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Carnot [Collet et al., 1991]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VIP-MDBMS [Kuhn and Ludwig, 1988]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object-Oriented (Frame-Based) Data Model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O*SQL [Litwin, 1992]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMS [Arenas and Knoblock, 1992]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pegasus [Ahmed et al., 1991]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functional Data Model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multibase [Landers and Rosenberg, 1982]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relational Data Model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LINDA [Wolski, 1989]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mermaid [Templeton et al., 1987]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MRDSM [Litwin and Abdellatif, 1987]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDS [Breitbart and Tieman, 1985]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

The Tight-Coupling Strategy

In the case of tightly-coupled systems, the detection of conflicts is performed by a system administrator and the actual resolution is accomplished by defining, a priori, one or more views which define the shared schemas for the system. A shared schema insulates the receiver from underlying data heterogeneity by providing a canonical representation of the data originating from disparate sources. Queries formulated against a shared schema can be transformed to subqueries which are submitted to component sources, and the results are translated to the canonical representation using the view definition. Early prototypes which have been constructed using the tight-coupling approach include Multibase [Landers and Rosenberg, 1982], ADDS [Breitbart and Tieman, 1985], and Mermaid [Templeton et al., 1987]. More recently, the same strategy has been employed for systems adopting object-oriented data models (e.g., Pegasus [Ahmed et al., 1991] based on the IRIS data model), frame-based knowledge representation languages (e.g., SIMS [Arens and Knoblock, 1992] using LOOM), as well as logic-based languages (e.g., Carnot [Collet et al., 1991] using CycL, an extension of first-order predicate calculus).

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

CREATE SUPERTYPE student OF student1, student2;

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

As we will point out in Section 2.3, this procedural encapsulation of conflicts present
a number of deficiencies from both the conceptual and operational point of view.

The Loose-Coupling Strategy

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

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

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

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

### 2.3 The Measure of a Viable Integration Strategy

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

The recent years, however, have witnessed an exponential growth in the number of sources and receivers which are demanding solutions to complex integration scenarios characterized by a large number of component systems operating in a diversified and dynamic environment\(^2\). This phenomenon can be attributed to a number of reasons\(^3\):

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

- the shift towards a client-server computing environment where information are managed in a decentralized manner in individual departments or functional units, as opposed to consolidating all information requirements of an organization in a large centralized database;

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

- the development and wide-spread acceptance of standards (e.g., URLs and the HTML mark-up language) and network protocols (e.g., the HTTP and IP protocols) which ease the transition to a networked environment; and

- the unprecedented growth in the number of web-sites that are connected to the World Wide Web which are acting as information publishers in varying capacities.

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

\(^3\)See also [Madnick, 1996] for elaboration of some of the reasons cited here.
Whatever the initial reasons may be, these have led to a “snowballing” effect. In economic terms, this is a classic scenario where positive network externalities is at play: as more and more organizations make their information resources and applications network-aware, the “marginal benefit” for the next source or receiver to come online becomes greater.

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

**Scalability**

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

First, the classical strategies adopted for semantic integration is generally not tenable when the number of sources and receivers is large. In the case of tightly-coupled systems, defining a shared schema encapsulating all of the underlying conflicts becomes prohibitively complex. This is not just because sources are more heterogeneous, but also due to the fact that receivers will most likely have more diverse requirements which must be accounted for in the design of the shared schema. (In some instances, this may require the creation of several shared schemas to cater to the needs of different users.) With loosely-coupled systems, receivers, instead of system administrators, bear the brunt of the work. On the whole, the problem is even more pronounced since end users have limited resources and hence may not be sufficiently equipped to deal with complex semantic issues.
From a query processing point of view, having greater diversity in component systems means that more data conversions are expected leading to the rapid degradation of system performance. To make matters worse, both the tight- and loose-coupling approaches provide little scope for query optimization since semantic conflicts are never represented explicitly but are encapsulated in the form of conversion functions (which cannot be easily inspected or reasoned with by a query optimizer). In a small and controlled environment, it is often possible for "canned" queries to be carefully hand-crafted to meet the needs of critical applications. Such an approach, however, becomes infeasible when a large number of receivers need to be supported and when frequent changes in the underlying system are expected.

**Extensibility**

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

Whenever system membership or underlying schemas/semantics change, the effects must somehow be reflected in the rest of the system which interoperates with the element that has been changed. In the case of tightly-coupled systems, this means that shared schemas referencing the altered component must be updated. In most instances, these changes must be manually performed by a system administrator. With loosely-coupled systems, such changes must be conveyed to receivers (having vested interest in them) who must somehow register the changes to facilitate future access. Even when such changes in individual systems occur infrequently, these could add up to formidable recurring events at the system level. For example, assuming an average of one change in three years for any component system, an integrated system with three hundred sources will have to contend with a hundred changes every year, translating to two changes every week! The reliance of classical strategies on manual interventions in mitigating these changes is clearly not a feasible choice.
Accessibility

The third criteria which we deem to be important is the ability to make information resources more accessible to receivers. There are at least two variations of this theme. First, making information more accessible means presenting information to a receiver in the representation that the latter expects. We have alluded to this earlier by pointing out that receivers, like sources, are heterogeneous and any integration strategy must seek to preserve their autonomy: i.e., receivers should be able to issue the same queries and receive answers in the same format/interpretation as they would normally expect (say, from a local data source). Although this can be accomplished in tightly-coupled systems through the use of shared schemas, this approach suffered from maintenance bottlenecks as mentioned earlier. In the case of loosely-coupled systems, the data is only accessible to the extent that the user is capable of deciphering the underlying semantic disparities. This is generally an untenable assumption.

Second, it is often useful for users to gain access to the meaning of data represented in different systems in addition to the extensional data. Intuitively, the ability to gain access to the implicit semantics of data present in disparate systems is important for a variety of reasons. For instance, a user may simply want to know differences in interpretation attributed to data reported in two different systems. At a different level, this information is also useful for purposes of query planning. For example, when multiple sources of the same data (having different representations in different systems) are available, knowledge of these differences can aid in selecting the source which are least costly (i.e., one that requires the minimal amount of data transformation). Despite these advantages, such a feature has not been reported in the database integration literature.

2.4 Discussion

The discussions in this chapter have provided both the background and the motivation for the Context Interchange approach which we will describe in this Thesis. For most part, we are concerned with the class of semantic conflicts as is described
in 2.1. This sometimes limits our ability to deal with certain types of conflicts. An example of such a conflict, which is illustrative of the "limits" of our integration strategy, can be found in Section 6.5. Finally, our discussion on "success measures" of an integration strategy has provided a forum for describing the relative merits of traditional loose- and tight-coupling approaches. A detailed comparison with more contemporary research projects and prototypes will be provided in Section 8.1.
"Every general theory must work in at least one case."
- Stuart E. Madnick

Chapter 3

Context Interchange By Example

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

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

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

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

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 3-1.

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

\(^1\)We assume, without loss of generality, that relation names are unique across all data sources. This can always be accomplished via some renaming scheme: say, by prefixing relation name with the name of the data source (e.g., db1#r1).
3.2 A User Perspective of Context Interchange

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

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

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

MQ1:  
SELECT r1.cname, r1.revenue FROM r1, r2  
WHERE r1.currency = 'USD' AND r1.cname = r2.cname  
AND r1.revenue > r2.expenses;

UNION

SELECT r1.cname, r1.revenue * 1000 * r3.rate FROM r1, r2, r3  
WHERE r1.currency = 'JPY' AND r1.cname = r2.cname  
AND r3.fromCur = r1.currency AND r3.toCur = 'USD'  
AND r1.revenue * 1000 * r3.rate > r2.expenses

UNION

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

This mediated query considers all potential conflicts between relations r1 and r2 when comparing values of “revenue” and “expenses” as reported in the two different contexts. Moreover, the answers returned may be further transformed so that they conform to the context of the receiver. Thus in our example, the revenue of NTT will be reported as 9 600 000 as opposed to 1 000 000. More specifically, the three-part query shown above can be understood as follows. The first subquery takes care of tuples for which revenue is reported in USD using scale-factor 1; in this case, there is
no conflict. The second subquery handles tuples for which revenue is reported in JPY, implying a scale-factor of 1000. Finally, the last subquery considers the case where the currency is neither JPY nor USD, in which case only currency conversion is needed. Conversion among different currencies is aided by the ancillary data source r3 which provides currency conversion rates. This second query, when executed, returns the “correct” answer consisting only of the tuple <’NTT’, 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

VQ1: \[
\text{SELECT cname, profit FROM v1}
\]

\[
\text{WHERE profit > 0;}
\]

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. This means that query mediation can be undertaken by the Context Mediator as before.
Knowledge-Level versus Data-Level Queries

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

\[
\text{Q2: } \quad \text{SELECT r1.cname, r1.revenue.scaleFactor IN c1,} \\
\quad \text{r1.revenue.scaleFactor IN c2 FROM r1} \\
\quad \text{WHERE r1.revenue.scaleFactor IN c1 <>} \\
\quad \text{r1.revenue.scaleFactor IN c2;}
\]

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

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

\[
\text{MQ2: } \quad \text{SELECT r1.cname, 1000, 1 FROM r1} \\
\quad \text{WHERE r1.currency = 'JPY';}
\]

\(^2\)Sciore et al. [1992] have described a similar (but not identical) extension of SQL in which contexts are treated as “first-class objects”. We are not concern with the exact syntax of such a language here; the issue at hand is how we might support the underlying inferences needed to answer such queries.
which indicates that the answer consists of companies for which the currency attribute has value 'JPY', in which case the scale-factors in context c1 and c2 are 1000 and 1 respectively. If desired, the mediated query MQ2 can be evaluated on the extensional data set to return an answer grounded in actual data elements. Hence, if MQ2 is evaluated on the data set shown in Figure 3-1, we would obtain the singleton answer <'NTT', 1000, 1>.

**Extensional versus Intensional Answers**

Yet another feature of our mediation approach is that answers to queries can be both intensional and extensional. Extensional answers correspond to fact-sets which one normally expects of a database retrieval. Intensional answers, on the other hand, provide only a characterization of the extensional answers without actually retrieving data from the data sources. In the preceding example, MQ2 can in fact be understood as an intensional answer for Q2, while the tuple obtained by the evaluation of MQ2 constitutes the extensional answer for Q2. In the COIN framework, intensional answers are grounded in extensional predicates (i.e., names of relations) as well as evaluable predicates (e.g., arithmetic operators, "relational" operators, and external functions which can be directly evaluated through system calls). The intensional answer is thus not substantively different from a query which can normally be evaluated on a conventional query subsystem of a DBMS. Query answering in a Context Interchange system is thus a two-step process: an intensional answer is first returned in response to a user query; this can then be executed on a conventional query subsystem to obtain the extensional answer.

The intermediary intensional answer serves a number of purposes [Imielinski, 1987]. Conceptually, it constitutes the mediated query corresponding to the user query and can be used to confirm the user's understanding of what the query actually entails. More often than not, the intensional answer can be more informative and easier to comprehend compared to the extensional answer it derives. (For example, the intensional answer MQ2 actually conveys more information than merely returning the single tuple satisfying the query.) From an operational standpoint, the
computation of extensional answers are likely to be many orders of magnitude more expensive compared to the evaluation of the corresponding intensional answer. It therefore makes good sense not to continue with query evaluation if the intensional answer satisfies the user. From a practical standpoint, this two-stage process allows us to separate query mediation from query optimization and execution. As we will illustrate later in this paper, query mediation is driven by logical inferences which do not bond well with the (predominantly cost-based) optimization techniques that have been developed [Mumick and Pirahesh, 1994, Seshadri et al., 1996]. The advantage of keeping the two tasks apart is thus not merely a conceptual convenience, but allows us to take advantage of mature techniques for query optimization in determining how best a query can be evaluated.

Query Pruning

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

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

**CONTEXT MEDIATION SERVICES**

![Diagram of Context Interchange system](image)

Figure 3-2: The architecture of a Context Interchange system.
Figure 3-2 depicts the architecture of a Context Interchange system. Observe that queries submitted by a receiver are not directly executed against the sources but are instead routed to a Context Mediator which transforms it into a mediated query. This mediation step is made possible by representing the semantics of data in disparate systems in the form of contexts associated with the sources, and with the aid of a domain model. In the remaining sections, we describe how this architecture is realized by focusing on its representational requirements.

**Representation of ‘Meaning’ as opposed to Conflicts**

As mentioned earlier, a key insight of the Context Interchange strategy is that we can represent the meaning of data in the underlying sources and receivers without identifying and reconciling all potential conflicts which exist between any two systems. Thus, query mediation can be performed dynamically (as when a query is submitted) or it can be used to produce a query plan (the mediated query) that constitutes a locally-defined view. In the latter case, this view is similar to the shared schemas in tightly-coupled systems with one important exception: whenever changes do occur (say, when the semantics of data encoded in some context is changed\(^3\)), the Context Mediator can be triggered to reconstruct the local view automatically\(^4\). 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.

**Decoupling Contexts from Schemas**

Figure 3-3 provides a graphical representation of the COIN framework, which embody the core representational components of the Context Interchange strategy. We

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

\(^4\)Although this may well be possible with tightly-coupled systems given the declarative nature of shared views, we have not been able to find any such discussion in the literature.
postpone the formal definition of this framework to Section 4.5 but offer an informal description of it below. Loosely speaking, the COIN framework consists of the following:

- a **domain model** which present 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;

- the **source set** which is the collection of facts in the sources and the relevant integrity constraints of each;

- the **elevation set** which consists of a set of axioms that create a **semantic-relation** corresponding to each extensional relation, and which defines the semantics of objects in this semantic-relation by relating them to types and their attributes in the domain model;

- a **context set**, consisting of a collection of context providing a succinct definition of the semantic of data with (ideally) no references to underlying schemas;

- a **hierarchical relation** (represented by \( \prec \)) between the collection of contexts in a context set; and

- a mapping (denoted by \( \mu \)) which identifies each data source with some context in the context set.

One of the best feature of this scheme is that it allows the semantics of a data element to be described at different levels of specificity independent of particular schemas or of how data is physically structured in the underlying system. Thus, the fact that all companyFinancials are reported in 'USD' in the context of \( c_2 \) can be represented in the clause:

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

In this instance, both the variables \( X' \) and \( \text{currency}(c_2, X') \) represents oid-terms which denote **semantic-objects**, which are instances of corresponding (semantic-)types
companyFinancials and semanticNumber defined in the domain model. The oid-term currency\((c_2, X')\) denotes the semantic-object corresponding to the currency of \(X'\) in context \(c_2\), and has a value 'USD' in context \(c_2\). The "method" currency\((c_2)\) constitute a modifier property of the semantic-type companyFinancials since it determines (in part) how one may interpret the value corresponding to a companyFinancials object. A detailed presentation of these features will be presented later in Section 4.3.

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

**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 (see Figure 3-4 for one such example), which allow us to take advantage of structural and behavioral inheritance in achieving economy of expression. *Structural inheritance* allows a semantic-type to inherit the declarations defined for its supertypes. For example, the semantic-type `companyFinancials` inherits from `moneyAmt` the declarations concerning the existence of the "methods" `currency` and `scaleFactor`. *Behavioral inheritance* allows the *definitions* of these methods to be inherited as well. Hence, if we had defined earlier that instances of `moneyAmt` has a scale-factor of 1, all instances of `companyFinancials` would inherit the same scale-factor since every instance of `companyFinancials` is an instance of `moneyAmt`.

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

**Value Conversion Among Different Contexts**

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

\[
X' : moneyAmt \leftarrow \]
Figure 3-4: 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.

\[ X'[\text{cvt}(c_1)@\text{scaleFactor}, C, U \rightarrow V] \leftarrow \]
\[ X'[\text{scaleFactor}(c_1) \rightarrow \ldots, \text{value}(c_1) \rightarrow F_1], \]
\[ X'[\text{scaleFactor}(C) \rightarrow \ldots, \text{value}(c_1) \rightarrow F], V = U \times F/F_1. \]

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

Hierarchical composition of Contexts

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

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

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

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'):

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

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

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

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

which distinguishes it as having a different \textit{format}. 
Chapter 4

The COIN Data Model

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

- a notation for describing data, and

- a set of operations used to manipulate that data.

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

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

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

Object-Oriented Data Models

In object-oriented data models [Zdonik and Maier, 1990], the “universe of discourse” is captured in the form of a collection of object templates called types or classes. Every real-world entity is represented by an object, which is an instance of some type. The type corresponding to an object defines a template for its structure, which includes both the data-structure and the behavioral aspects of that object. Each object is identified by a unique object identifier (oid), which serves as the basis for sharing and the construction of complex objects. Every object has a state defined via a set of properties, which can be queried or operated upon through the use of methods. This feature is referred to as encapsulation.
Methods can be *functional* (i.e., single-valued) or *multi-valued*\(^1\). A functional method returns only one value when the method is applied to an object; a multi-valued method returns one or more values. The definition of a method has two components: *signature* and *body*. The signature specifies the name of the method, the number and types of arguments it expects, and the type(s) of the result. The body represents the implementation of the method, which can be realized by a sequence of instructions written in some programming languages.

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

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

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

\(^1\)They could also be "procedural", in which case the methods are used for achieving some side-effects and do not return any meaningful value.
Deductive Data Models

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

\[ L_0 \leftarrow L_1, \ldots, L_n, \quad n \geq 0 \]

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

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

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

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

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

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

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

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

**Deductive Object-Oriented Data Models**

As noted earlier, the interest behind deductive object-oriented data models is largely a response to inadequacies of object-oriented data models and deductive data models. Despite this common endeavor, these efforts vary somewhat depending on the research traditions in which they are grounded. For example, both LOGIN [Aït-Kaci and Nasr, 1986] and $i^2$O [McCabe, 1992] are in fact object-oriented extensions to classical logic programming languages [Lloyd, 1987]. On the other hand, proposals such as $O$-logic [Maier, 1986, Kifer and Wu, 1989], $C$-logic [Chen and Warren, 1989], $F$-logic [Kifer and Lausen, 1989, Kifer et al., 1995] and Gulog [Dobbie and Topor,
earning more than 2000 are required to pay taxes (proportional to the salaries they earn), we may write

\[ X:\text{working-student} \leftarrow X[\text{tax} \rightarrow T] \leftarrow X[\text{salary} \rightarrow S] \land S > 2000 \]
\[ \land T = 0.1 \times S. \]

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

\begin{align*}
\text{jack} : & \text{working-student.} & \text{jack}[\text{salary} \rightarrow 3000]. \\
\text{jill} : & \text{working-student.} & \text{jill}[\text{salary} \rightarrow 1500].
\end{align*}

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

### 4.2 Formalizing Context for Semantic Interoperability

McCarthy [1987] 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:

\[ c : \text{ist}(c, \sigma) \]

which suggests that the statement \( \sigma \) is true ("ist") in the context \( c \), this statement itself being asserted in an outer context \( \bar{c} \). Lifting axioms\(^2\) are used to describe the relationship between statements in different contexts. These statements are of the form

---

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

\(^3\)also called articulation axioms in Cyc/Carnot [Collet et al., 1991].
\( \bar{c} : \ ist(c, \sigma) \leftrightarrow ist(c', \sigma') \)

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 may be false in a different context). Thus, if we assign the labels \( c_1 \) and \( c_2 \) to contexts associated with sources 1 and 2 in Figure 3-1, we may now write:

\[
\begin{align*}
\bar{c} : & \ ist(c_1, r1('NTT', 1000000, 'JPY')). \\
\bar{c} : & \ ist(c_2, r2('NTT', 5000000)).
\end{align*}
\]

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

In the Context Interchange approach, the semantics of data are captured explicitly in a collection of statements asserted in the context associated with each source, while allowing conflict detection and reconciliation to be deferred to the time when a query is submitted. Building on the ideas developed in [Siegel and Madnick, 1991, Sciore et al., 1994], we would like to be able to represent the semantics of data at the level of individual data elements (as opposed to the predicate or sentential level), which allows us to identify and deal with conflicts at a finer level of granularity. Unfortunately, individual data elements may be present in a relation without a unique denotation. For instance, the value 1000000 in relation \( r1 \) (as shown in Figure 3-1) simultaneously describes the revenue of IBM and NTT while being reported in different currencies and scale-factors. Thus, the statements

\[
\begin{align*}
ist(c_1, currency(R, Y) & \leftarrow r1(N, R, Y)). \\
ist(c_1, scaleFactor(R, 1000) & \leftarrow currency(R, Y), Y = 'JPY'). \\
ist(c_1, scaleFactor(R, 1) & \leftarrow currency(R, Y), Y \neq 'JPY').
\end{align*}
\]
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. As we shall see later, semantic-objects are complex terms constructed from the corresponding data values (also called primitive-objects) and are used as a basis for inferring about conflicts, but are never materialized in an object-store.

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

4.3 The Structural Elements of COIN

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

Primitive-types correspond to data types (e.g., strings, integers, and reals) which are native to sources and receivers. Semantic-types, on the other hand, are complex types introduced to support the underlying integration strategy. Specifically, semantic-objects may have properties which are either attributes or modifiers. At-
tributes 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 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.

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

\[(1) \ \text{ist}(c_1, \text{value}(sk_0, 1\,000\,000)); \text{ and} \]
\[(2) \ \text{ist}(c_2, \text{value}(sk_0, 9\,600\,000)), \]

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

**Definition 1** 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

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

We refer to $f_{cvt}$ as the 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 $c_s$. \(\square\)

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

In many instances, the modifiers of a semantic-type are "orthogonal", in the sense that the value which is assigned to one is independent of others (e.g., scaleFactor and currency). Under these circumstances, the conversion function referenced in the preceding definition can in fact be composed from other simpler conversion methods defined with reference to each modifier. To distinguish between the two, we refer to the first as a composite conversion function, and the latter as atomic conversion functions. Suppose $m_1, \ldots, m_k$ are orthogonal modifiers of a semantic-type $\tau$, 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; otherwise they are oid-equivalent. 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 for any context $c$,

$$ist(c, (t \oplus t')) \Leftrightarrow (value(t, X) \land value(t', X') \land X \oplus X')$$

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

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

\[
sk_0 \approx 5\,000\,000
\]

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

### 4.4 The Language of COIN

We describe in this section the syntax and informal semantics of the language of COIN, which is inspired largely by \textit{Gulog} [Dobbie and Topor, 1995]. Rather than making inferences using a context logic (see, for example, [Buvač, 1995]), we introduce "context" as first-class objects and capture variations in different contexts through the use of parameterized methods. For example, the context-formula \( ist(c_1, value(sk_0, 1\,000\,000)) \) can be equivalently written as \( sk_0[value(c_1) \rightarrow 1\,000\,000] \) where \( value(c_1) \) represents a (single-valued) method; similarly, we will write \( X \bowtie Y \) in place of \( ist(c, X \bowtie Y) \). 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 [Lloyd, 1987], we define an alphabet as consisting of (1) a set of type symbols which are partitioned into symbols representing semantic-types and primitive-types: each of which have a distinguished type symbol, denoted by \( T_S \) 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., value and cvt); (5) an infinite set of variables; (6) the usual logical connectives and quantifiers ∨, ∀, ∃, ¬, etc; (7) auxiliary symbols such as (, ), [, ], :, ::, →, ⇒ 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(t₁,...,tₙ) where f is a function symbol and t₁,...,tₙ are terms. Since terms in our model refer to (logical) oids, they are called oid-terms. Finally, a predicate, function, or method symbol is said to be n-ary if it expects n arguments.

**Definition 3** A declaration is defined as follows:

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

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

- if p is an n-ary predicate symbol, and τ₁,...,τₙ are type symbols, then p(τ₁,...,τₙ) is a predicate declaration. We say that the signature of predicate p is τ₁ × ··· × τₙ.

- if m is an attribute symbol and τ, τ' are symbols denoting semantic-types, then τ[m⇒τ'] is an attribute declaration. We say that the signature of the attribute is τ→τ', and that the semantic-type τ has attribute m.

- if m is a modifier symbol, and τ, τ' are symbols denoting semantic-types, then τ[m(CTX)⇒τ'] is a modifier declaration. We say that m is a modifier of the semantic-type τ, which has signature τ→τ'.

- if τ is a semantic-type, and τ₁, τ₂ are primitive types, then τ[cvt(CTX)@CTX,τ₁⇒τ₂] is a compound conversion declaration. We say that the signature of the compound conversion for τ is τ × τ₁→τ₂.
• if $\tau$ is a semantic-type, $m$ is a modifier defined on $\tau$, and $\tau_1, \tau_2$ are primitive types, then $\tau[\text{cvt}(\text{ctx})@m, \text{ctx}, \tau_1 \Rightarrow \tau_2]$ is a 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 $\tau[\text{value}(\text{ctx}) \rightarrow \tau_1]$ is a 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 method declarations.

Definition 4 An atom is defined as follows:

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

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

• if $m$ is a modifier symbol with signature $\tau \rightarrow \tau'$, $c$ is a context symbol, and $t, t'$ are of (inherited) types $\tau, \tau'$ respectively, then $t[m(c) \rightarrow t']$ is a 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 $t[\text{cvt}(c)@t_c, t_1 \rightarrow t_2]$ is a compound conversion atom.

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

• if the value signature is given by $\tau \rightarrow \tau'$, $c$ is a context symbol, and $t, t'$ are of (inherited) types $\tau, \tau_1$, then $t[\text{value}(c) \rightarrow t']$ is a value atom.

As before, the atoms corresponding to attributes, modifiers, conversions, and built-in method value are referred to collectively as method atoms.
Atoms can be combined to form molecules (or compound atoms): these are “syntactic sugar” which are notationally convenient, but by themselves do not increase the expressive power of the language. For example, we may write

- \( t[m_1 \rightarrow t_1; \cdots; m_k \rightarrow t_k] \) as a shorthand for the conjunct \( t[m_1 \rightarrow t_1] \land \cdots \land t[m_k \rightarrow t_k] \);

- \( t[m \rightarrow t][m_1 \rightarrow t_2] \) as a shorthand for \( t[m \rightarrow t_1] \land t_1[m_1 \rightarrow t_2] \); and

- \( t : \tau[m \rightarrow t'] \) as a shorthand for \( t : \tau \land t[m \rightarrow t'] \).

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

- an atom is a formula;

- if \( \phi \) 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. \end{quote}

4.5 The COIN Framework

The COIN framework builds on the COIN data model to provide a formal characterization of the Context Interchange strategy for the integration of heterogeneous data sources.
Definition 6 A COIN framework \( \mathcal{F} \) is a quintuple \( \langle \mathcal{S}, \mu, \mathcal{E}, \mathcal{D}, \mathcal{C} \rangle \) where

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

- \( \mu \), the source-to-context mapping, defines a (total) function from \( \mathcal{S} \) to \( \mathcal{C} \). If \( \mu(s_i) = c_j \), we say that the source \( s_i \) is in context \( c_j \). If \( r \) is a relation in \( s_i \), we sometimes abuse the notation and write \( \mu(r) = c_j \) instead.

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

- \( \mathcal{E} \), the elevation set, is a multi-set \( \{ E_1, \ldots, E_m \} \) where \( E_i \) is the set of elevation axioms corresponding to \( s_i \) in \( \mathcal{S} \). \( E_i \) consists of three parts:
  - for each relation \( r_{ij} \in S_i \), there is a clause which defines a corresponding semantic-relation \( r'_{ij} \) in which every primitive object in \( r_{ij} \) is replaced by a Skolem term in \( r'_{ij} \);
  - for every oid-term in \( r'_{ij} \), we define its value in context \( c(= \mu(s_i)) \) with reference to \( r_{ij} \); and
  - 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).

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

We provide the intuition for the above definition by demonstrating how the integration scenario shown in Figure 3-1 can be represented in a COIN framework.
Source set $S$

$$s_1 := \{ r_1(‘IBM’, 1\ 000\ 000, ‘USD’), r_1(‘NTT’, 1\ 000\ 000, ‘JPY’).$$

$$R_1 = R_2 \leftarrow r_1(N, R_1), r_1(N, R_2).$$

$$Y_1 = Y_2 \leftarrow r_1(N, Y_1), r_1(N, Y_2). \}$$

$$s_2 := \{ r_2(‘IBM’, 1\ 500\ 000), r_2(‘NTT’, 5\ 000\ 000).$$

$$E_1 = E_2 \leftarrow r_2(N, E_1), r_2(N, E_2). \}$$

$$s_3 := \{ r_3(‘USD’, ‘JPY’, 104.0), r_3(‘JPY’, ‘USD’, 0.0096).$$

$$T_1 = T_2 \leftarrow r_3(X, Y, T_1), r_3(X, Y, T_2). \}$$

Source-to-Context Mapping $\mu$

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

Domain model $D$

```plaintext
/* type declarations */
semanticNumber :: $T_S$. number :: $T_P$.
semanticString :: $T_S$. varchar :: $T_P$.
moneyAmt :: semanticNumber. integer :: number.
companyFinancials :: moneyAmt. real :: number.
exchangeRate :: semanticNumber.
currencyType :: semanticString.
companyName :: semanticString.

/* attribute declaration */
companyFinancials[company \Rightarrow companyName].
exchangeRate[fromCur \Rightarrow currencyType; toCur \Rightarrow currencyType].

/* modifier declarations */
moneyAmt[currency(ctx)\Rightarrow currencyType; scaleFactor(ctx)\Rightarrow semanticNumber].

/* value declarations */
semanticString[value(ctx)\Rightarrow varchar].
semanticNumber[value(ctx)\Rightarrow number].

/* conversion declarations */
semanticString[cvt(ctx)\@ctx, varchar \Rightarrow varchar].
semanticNumber[cvt(ctx)\@ctx, number \Rightarrow number].
moneyAmt[cvt(ctx)\@ctx, number \Rightarrow number].
moneyAmt[cvt(ctx)\@ctx, scaleFactor, number \Rightarrow number].
moneyAmt[cvt(ctx)\@ctx, currency, number \Rightarrow number].

/* predicate declarations */
r_1(companyName, companyFinancials, currencyType). r_1(varchar, integer, varchar).
r_2(companyName, companyFinancials). r_2(varchar, integer).
r_3(currencyType, currencyType, semanticNumber). r_3(varchar, varchar, real).
```

Figure 4-1: The source set, source-to-context mapping, and domain model for the COIN framework corresponding to the motivational example.
\[ \mathcal{F} = \langle S, \mu, \mathcal{E}, D, C \rangle. \] Figures 4-1 and 4-2 present a partial codification which we will elaborate briefly below:

- The contents of the source set \( S \) is simply the set of ground atoms and integrity constraints 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 \( \text{cname} \rightarrow \{ \text{revenue}, \text{currency} \} \) 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 \( c_2 \).

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

- 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., \( f_{r2\#\text{expenses}} \)) are chosen in the way such that when applied to the key-value of a tuple in the corresponding relation (e.g., \( \text{'NTT'} \)), the resulting Skolem term (i.e., \( f_{r2\#\text{expenses}}(\text{'NTT'}) \)) would in fact identify a unique "cell" in the relation as shown in Figure 3-4 (in this case, the expenses of NTT as reported in relation \( r_2 \)).

- 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.

- Object declarations and attribute atoms in the elevation set provide a way of specifying the types of corresponding Skolem terms introduced in the semantic relation. For instance, any Skolem term \( f_{\text{r}1\#\text{revenue}(\cdot)} \) is asserted to be an instance of the semantic-type \textit{companyFinancials}. The attribute atom following this declaration defines the object that is assigned to the \textit{company} attribute for this semantic-object.

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 4-3. There are two kinds of axioms: modifier value definitions and conversion definitions.

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

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

\begin{align*}
(X'[\text{scaleFactor}(c_1) \rightarrow F']) \land \\
(F'[\text{value}(c_1) \rightarrow 1000] & \leftarrow X'[\text{currency}(c_1) \rightarrow Y'] \land Y' \overset{c_1}{=} \text{JPY}') \land \\
F'[\text{value}(c_1) \rightarrow 1] & \leftarrow X'[\text{currency}(c_1) \rightarrow Y'] \land Y' \not\overset{c_1}{=} \text{JPY}'.
\end{align*}
\]

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\(^5\):

\[
X' : \textit{companyFinancials} \vdash
\]

\(^5\)The first clause introduces a Skolem object which is referenced by the subsequent two. In general, this could have been created through a meta-logical assertion and need not be explicitly asserted. We have chosen to make this explicit to illustrate the rationale underlying the choice of oids for the Skolem objects.
Elevation Axioms $E_1$ of $\mathcal{E}$

$$r'_1(f_1\#\text{name}(X_1), f_1\#\text{revenue}(X_1), f_1\#\text{currency}(X_1)) \leftarrow r_1(X_1, \ldots).$$

$f_1\#\text{name}(\ldots) : \text{company\text{-}Name}.$

$f_1\#\text{revenue}(\ldots) : \text{company\text{-}Financials}.$

$f_1\#\text{currency}(\ldots) : \text{currency\text{-}Type}.$

$f_1\#\text{name}(X_1)[\text{value}(C) \rightarrow X_1] \leftarrow r_1(X_1, \ldots), \mu(s_1, C).$

$f_1\#\text{revenue}(X_1)[\text{value}(C) \rightarrow X_2] \leftarrow r_1(X_1, X_2, \ldots), \mu(s_1, C).$

$f_1\#\text{currency}(X_1)[\text{value}(C) \rightarrow X_3] \leftarrow r_1(X_1, \ldots, X_3), \mu(s_1, C).$

$R' \triangleright \text{company\text{-}Financials} \vdash R'[\text{company} \rightarrow N'] \leftarrow r'_1(N', R', \ldots).$

Elevation Axioms $E_2$ of $\mathcal{E}$

$$r'_2(f_2\#\text{name}(X_1), f_2\#\text{expenses}(X_1)) \leftarrow r_2(X_1, \ldots).$$

$f_2\#\text{name}(\ldots) : \text{company\text{-}Name}.$

$f_2\#\text{expenses}(\ldots) : \text{company\text{-}Financials}.$

$f_2\#\text{name}(X_1)[\text{value}(C) \rightarrow X_1] \leftarrow r_2(X_1, \ldots), \mu(s_2, C).$

$f_2\#\text{expenses}(X_1)[\text{value}(C) \rightarrow X_2] \leftarrow r_2(X_1, X_2, \ldots), \mu(s_2, C).$

$R' \triangleright \text{company\text{-}Financials} \vdash R'[\text{company} \rightarrow N'] \leftarrow r'_2(N', R').$

Elevation Axioms $E_3$ of $\mathcal{E}$

$$r'_3(f_3\#\text{from\text{-}Cur}(X_1, X_2), f_3\#\text{to\text{-}Cur}(X_1, X_2), f_3\#\text{exchange\text{-}Rate}(X_1, X_2)) \leftarrow r_3(X_1, X_2, \ldots).$$

$f_3\#\text{from\text{-}Cur}(\ldots, \ldots) : \text{currency\text{-}Type}.$

$f_3\#\text{to\text{-}Cur}(\ldots, \ldots) : \text{currency\text{-}Type}.$

$f_3\#\text{exchange\text{-}Rate}(\ldots, \ldots) : \text{semantic\text{-}Number}.$

$f_3\#\text{from\text{-}Cur}(X_1, X_2)[\text{value}(C) \rightarrow X_1] \leftarrow r_3(X_1, X_2, \ldots), \mu(s_3, C).$

$f_3\#\text{to\text{-}Cur}(X_1, X_2)[\text{value}(C) \rightarrow X_2] \leftarrow r_3(X_1, X_2, \ldots), \mu(s_3, C).$

$f_3\#\text{exchange\text{-}Rate}(X_1, X_2)[\text{value}(C) \rightarrow X_3] \leftarrow r_3(X_1, X_2, X_3, \ldots), \mu(s_3, C).$

$X' \triangleright \text{exchange\text{-}Rate} \vdash X'[\text{from\text{-}Cur} \rightarrow F'; \text{to\text{-}Cur} \rightarrow T'] \leftarrow r'_3(F', T', X').$

Figure 4-2: Elevation set corresponding to the motivational example
Conte. 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' \notin \{JPY'\}.
X' : companyFinancials, scaleFactor(c1, X') : number ⊨
    scaleFactor(c1, X')[value(c1) → 1 000] ← X'[currency(c1) → Y'], Y' \notin \{JPY'\}.
X' : companyFinancials ⊨ X'[currency(c1) → currency(c1, X')].
X' : companyFinancials, currency(c1, X') : currencyType ⊨
    currency(c1, X')[value(c1) → Y] ← X'[company → N_0'], r'_{(N_0', R', Y')}, N_0' \equiv N_1',
    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/F.
X' : moneyAmt ⊨
    X'[cvt(c1)@currency, C, U → V] ←
    X'[currency(c1) → Y'], X'[currency(C) → Y'],
    Y' \equiv Y', V = U.
X' : moneyAmt ⊨
    X'[cvt(c1)@currency, C, U → V] ←
    X'[currency(c1) → Y'], X'[currency(C) → Y'],
    Y' \equiv Y', R' : exchangeRate[fromCur → Y'_i; toCur → Y'_i],
    Y'_i \equiv Y', Y'_i \equiv Y'_i,
    R'[value(c1) → R], 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 4-3: Context sets for C for the motivational example at hand.
\( X'[\text{scaleFactor}(c_1) \rightarrow f_{\text{scaleFactor}(c_1)}(X')] \).

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

\( X' : \text{companyFinancials}, f_{\text{scaleFactor}(c_1)}(X') : \text{number} \leftarrow X'[\text{currency}(c_1) \rightarrow Y'], Y' \overset{c_3}{\not\leftrightarrow} \text{JPY} \).

where \( f_{\text{scaleFactor}(c_1)} \) is a unique Skolem function; for notational simplicity, we sometimes replace \( f_{\text{scaleFactor}(c_2)}(X') \) with the term \( \text{scaleFactor}(c_2, X') \). Currency values corresponding to instances of \textit{companyFinancials} are obtained directly from the extensional relation \( r_1 \) as shown in Figure 4-3. In this instance, it is necessary to reference an extensional relation because "meta-data" are represented along with "data" in a source. In a "better-behaved" situation (such as context \( c_2 \)), the modifier values for \textit{currency} and \textit{scaleFactor} can be defined independently of the underlying schema.

It is worthwhile to note that our framework is sufficiently expressive to capture both types of scenario, although the first tends to make the boundary between intensional and extensional knowledge more fuzzy.

Conversion functions define how the value of a given semantic-object can be derived in the current context, given that its value is known with respect to a different context. As shown in Figure 4-3, the first clause in the group (for context \( c_1 \)) defines the conversion for \textit{moneyAmt} via the composition of atomic conversion functions for \textit{scaleFactor} and \textit{currency}. The \textit{scaleFactor} conversion is defined by identifying the respective scale-factors in the source and target contexts and multiplying the value of the \textit{moneyAmt} object by the ratio of the two. The \textit{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 \textit{moneyAmt} but are applicable to \textit{companyFinancials} via behavioral inheritance of the methods. In general, the repertoire of conversion functions can be extended arbitrarily by invoking functions which are externally defined. 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.

4.6 A Meta-Logical Extension to the COIN Framework

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

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

\[
ist(c_i, \sigma)
\]

is taken to mean that the statement \( \sigma \) 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:

\[
\begin{align*}
ist(c_i, \sigma) & \leftarrow \sigma \in \delta_{c_i} \\
ist(c_i, \sigma) & \leftarrow c_i \prec c_j, \ ist(c_j, \sigma), \ not-overridden(\delta_{c_i}, \sigma).
\end{align*}
\]

The predicate not-overridden indicates that the statement \( \sigma \) obtained from the more general context \( c_j \) is not explicitly overridden by the differential context. The com-

\footnote{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 [Darwala et al., 1995].}
position of a new context theory of $c_i$ from $c_j$ and $\delta_{c_i}$ is similar to that accomplished by the $isa$ operator defined in [Borgi et al., 1990].

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

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

can be equivalently stated as

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

This second form simplifies the inferences which are undertaken to support context mediation, but requires some adjustment to allow statements to be inherited. Specifically, if the above statement is inherited by context $c_i \prec c_j$, we will need to replace the references to $c_j$ with $c_i$. This is accomplished by requiring all statements in $\delta_{c_j}$ 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 7** 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 $\{c_{i_1}, \ldots, c_{i_k}\}$ be the predecessors of $c_i$ with respect to the subcontext relation $\prec$. Then the uninstantiated context set for $c_{i_j}$, denoted by $C_{i_j}(X)$, can be obtained from $C_i(X)$ as before: i.e.,

- $\sigma(X) \in C_{i_j}(X) \leftarrow \sigma(X) \in \delta_{c_{i_j}}(X)$

- $\sigma(X) \in C_{i_j}(X) \leftarrow \sigma(X) \in C_i(X), not-overridden(\delta_{c_{i_j}}(X), \sigma(X)).$
The context \( c_0 \) is said to be the *default context* and forms the basis for the other differentials.

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

Notice that we have not described how one is to determine whether or not a given statement is being overridden in a specific context. One approach is to allow only static over-riding such 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 super-context applies. For example, if the `scaleFactor` for the type `companyFinancials` is given in two distinct context differentials along a given path in the hierarchy, then the statement in the more specific context is said to take precedence and will be used in the corresponding context.

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

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

In the above framework, the context labeled \( c_0 \) is said to be the *default context* for the integration environment. Since \( c_0 \) is the top-most context in the hierarchy, the values assigned to modifiers and conversion functions will be inherited by all other contexts unless they are being explicitly overridden. As we have described in Section 3.3, this provides a useful mechanism for evolving the domain model without the need to modify every context which has been previously defined. The use of this default context also means that a source or receiver wishing to participate in this environment need not undertake the laborious process of defining the context axioms corresponding to all the semantic-types present in the domain model. Instead, the
newly introduced source or receiver will simply inherit a context which is most like itself and incrementally refine the latter by introducing new axioms that overrides the ones that are inherited.
Chapter 5

Query Answering in the COIN Framework

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

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

\textsuperscript{2}This change in perspective is beneficial for a variety of reasons (see Chapter 3), and will not be repeated here.
5.1 Abductive Logic Programming

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

From observing \( A \) and the axiom \( B \rightarrow A \)

Infer \( B \) as a possible “explanation” of \( A \).

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

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

\[
(1) \quad \mathcal{T} \cup \Delta \models q(\bar{X}) \theta, \\
(2) \quad \mathcal{T} \cup \Delta \text{ satisfies } \mathcal{I}; \text{ and} \\
(3) \quad \Delta \text{ has some properties that make it “interesting”}. 
\]

This characterization of abduction is independent of the language in which the sentences are formulated. Requirement (1) states that \( \Delta \), together with \( \mathcal{T} \), must be
capable of providing an explanation for the observation $q(\bar{X})\theta$. 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.

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

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

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

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

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

$$
\begin{align*}
\Delta_1 &= \{ \text{punctured-tube } \} \\
\Delta_2 &= \{ \text{leaky-valve } \} \\
\Delta_3 &= \{ \text{broken-spokes } \}
\end{align*}
$$

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

$$
\begin{align*}
\mathcal{T}' &= \{ \text{flat-tire } \lor \text{broken-spokes } \leftarrow \text{wobbly-wheel.} \\
&\quad \text{punctured-tube } \lor \text{leaky-valve } \leftarrow \text{flat-tire.} \\
&\quad \text{wobbly-wheel } \leftarrow . \}
\end{align*}
$$

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

$$
\begin{align*}
M_1 &= \{ \text{wobbly-wheel, flat-tire, punctured-tube } \} \\
M_2 &= \{ \text{wobbly-wheel, flat-tire, leaky-valve } \} \\
M_3 &= \{ \text{wobbly-wheel, broken-spokes } \}
\end{align*}
$$

These corresponds exactly to the abductive solutions obtained earlier if they are restricted to only abducible predicates. As was pointed out in Console et al. [1991],
and subsequently Denecker and Schreye [1992a], the duality between abduction and
deduction holds in the general case and not just only when the theory is propositional
(i.e., variable-free). Denecker [1993] went further to point out that when a theory is
“complete” (in the sense that every formula or its negation is a logical consequence
of the theory), abduction then collapses to deduction.

5.3 The SLD+Abduction Procedure and Its Ex-
tensions

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

We first consider $SLD$ resolution\(^3\). Given a theory $T$ consisting of Horn clauses
and a goal clause $\leftarrow G$, and $SLD$-refutation of $\leftarrow G$ is a sequence of goal clauses
$\leftarrow G_0(= G); \leftarrow G_1; \cdots; \leftarrow G_n$ where $\leftarrow G_n$ is the empty clause (\Box) 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 $T$. The Prolog interpreter, for instance, implements a form of
SLD resolution where the leftmost literal in a goal clause is always selected. Since
there may be many clauses in $T$ which can be resolved with the selected literal, a
space of possible refutations is defined (in the form of an $SLD$-tree). The search
space defined by an SLD-tree may be searched in a number of ways. For example,
this is accomplished in Prolog in a depth-first manner, resulting in chronological
backtracking.

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

\(^3\)SLD resolution stands for SL resolution for Definite clauses. SL stands for Linear resolution
with Selection function. The term $LUSH$-resolution is sometimes used to refer to the same thing.
In the description which follows, we assume some familiarity with logic programming at the level
of [Lloyd, 1987, Chapters 1 and 2].
to a refutation (i.e., one which terminates in an empty clause). Given however that we are searching for a set of unit clauses $\Delta$, 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 $<T, A, I>$ and the abductive query $q(\bar{X})$, consider the sequence given by

\[ \leftarrow G_0, \Delta_0 \text{ where } G_0 = q(\bar{X}) \text{ and } \Delta_0 \text{ is the empty set} \]
\[ \vdots \]
\[ \leftarrow G_n, \Delta_n \]

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 $T \cup \Delta_i \cup \{g' \leftarrow\}$ is consistent with $I$, 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 $<T, A, I>$. A derivation, as we have just defined, is said to be a refutation if $\leftarrow G_n$ is the empty clause. The accumulated set of unit clauses $\Delta_n$ is said to be the residue corresponding to this refutation, and constitutes the abductive answer to $q(\bar{X})\theta$, 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 Skolem-ized. This is because variables in the unit clause "$g' \leftarrow" needs only be existentially quantified for it to be resolvable with $g$. If the Skolemization is not done, the abducted fact "$g' \leftarrow" (where $g' = g$) would have been unnecessarily strong. This Skolemization, however, introduces additional complexity since it becomes necessary to deal with equality constraints on Skolem constants. This is due to the fact that a Skolem
constant \(sk\) introduced earlier in the SLD-derivation (say in \(\leftarrow G_i\)) may have to be unified with a specific term \(t\) later on (in \(\leftarrow G_j\), where \(j > i\)). Eshghi [1988] suggested that this can be dealt with by introducing the equality predicate as an abducible predicate and to add the theory of Free Equality (FEQ) [Clark, 1978] as integrity constraints. Thus, when a Skolem constant \(sk\) is to be unified with a term \(t\), the equality fact \(sk = t\) is abducted explicitly and the consistency of \(sk = t\) with other abducted facts and FEQ is checked.

The procedure which we have just described can be extended to cope with negation through the use of negation-as-failure [Eshghi and Kowalski, 1989]. Suppose that the selected literal of the current goal clause is \(\neg g\). The usual negation-as-failure mechanism is used: i.e., if \(g\) cannot be proven from the theory (augmented with the current residue), then \(\neg g\) 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, \(\neg g\) 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 \(g \leftarrow \neg h\), and that \(h\) is not provable from the current residue. Then an attempt to prove \(\neg g\) using SLD-resolution with negation-as-failure (SLDNF) will fail because it is not possible to prove \(h\). However, \(h\) might be rendered provable by adding further clauses to the residue. So rather than using SLD-resolution to try to show \(h\), abduction is used instead and is allowed to add to the residue. This procedure can be generalized to any level of nesting, with SLD being used at even levels, and abduction at odd levels.

### 5.4 Query Answering in the COIN Framework

Figure 5-1 illustrates how queries are evaluated in a Context Interchange system. From a user perspective, queries and answers are couched in the relational data model: a (data-level or knowledge-level) query is formulated using a relational query language (SQL or some extension thereof), and answers can either be intensional (a mediated
query) or extensional (actual tuples satisfying the query). Examples of these queries
and answers have been presented earlier in Section 3.2.

Figure 5-1: A summary of how queries are processed within the Context Interchange
strategy: ① transforms a (extended) SQL query to a well-formed COIN query; ②
performs the COIN to Datalog\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
Chapter 3 can be mapped to the following clausal representations:

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

\[
\text{answer}(N, R) \leftarrow r_1(N, R, -), r_2(N, E), R > E
\]
and correspondingly,

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

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

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

- replace all relational operators with their "semantic" counterpart; for example, \( X > Y \) is replaced with \( X \overset{c}{\geq} Y \).

- make all relational "joins" explicit by replacing shared variables with explicit equality using the semantic-operator \( \overset{c}{=} \); for example, \( r_1(X, Y), r_2(X, Z) \) would be replaced with \( r_1(X_1, Y), r_2(X_2, Z), X_1 \overset{c}{=} X_2 \).

- similarly, make relational "selections" explicit; thus, \( r_1(X, a) \) will be replaced by \( r_1(X, Y), Y \overset{c}{=} a \).

- replace all references to extensional relations with the corresponding semantic-relations; for example, \( r_1(X, Y) \) will be replaced with \( r'_1(X, Y) \).

- append to the query constructed so far, value atoms that return the value of the data elements that are requested in the query.

Based on the above transformation, the well-formed query corresponding to naive queries \( <CQ1, c_2> \) and \( <CQ2, c_2> \), are given by
CQ1': \( \leftarrow \text{answer}(N, R) \).

\[
\text{answer}(N, R) \leftarrow r'_1(N'_1, R', \_), r'_2(N'_2, E'), N' \stackrel{c_2}{\equiv} N'_2, R' \stackrel{c_2}{\triangleright} E',
\]

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

and

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

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

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

respectively.

In this translation, all semantic-comparisons are performed in the context of the user issuing the query. For example, in comparing two company names originating from two distinct sources, the names are said to be identical if their values in the context of the user are identical. This provides a more natural interpretation of semantic-equivalence compared to that described in [Sciore et al., 1994], where semantic-equivalence is contingent on equivalence in any context after applying the conversion functions associated with all “meta-attributes” common to the two semantic-value being compared.

Transformations to an Abductive Framework

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

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

- \( T \) is the Datalog\(^{neg} \) translation of the set of clauses given by \( \mathcal{E} \cup \mathcal{D} \cup \mathcal{C} \cup \mu \);

- \( I \) consists of the integrity constraints defined in \( S \), augmented with Clark’s Free Equality Axioms [Clark, 1978]; and
• \( A \) consists of the extensional predicates defined in \( S \), the built-in predicates corresponding to arithmetic and relational (comparison) operators, and predicates corresponding to externally defined functions.

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 \( \mathcal{F}_A = <T, I, A> \). 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\textsuperscript{neg} is a sub-language of COIN, and any COIN query \( \leftarrow Q(\overline{X}) \) can always be transformed to a Datalog\textsuperscript{neg} query \( \leftarrow q(\overline{X}) \) by adding the Datalog\textsuperscript{neg}-translation of the clause \( q(\overline{X}) \leftarrow Q(\overline{X}) \) into the theory \( T \).

Given an abductive framework \( <T, I, A> \), 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

\[
T \models (q(\overline{X})\theta \leftarrow p_1, \ldots, p_m)
\]

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 \( q(\overline{X})\theta \). 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 \( \overline{X} \). Then, we say that the tuple \( (\exists Y (p_1, \ldots, p_m) \varphi, \theta \varphi) \) is an intensional answer for the query \( \exists \overline{X} q(\overline{X})(\theta \varphi) \). This fact is not surprising given that Motro and Yuan [1990] suggested that intensional answers can be obtained from the "dead-ends" of "derivation trees" corresponding to a query. Although it was not recognized as such, the procedure described in [Motro and Yuan, 1990] is in fact a naive implementation of SLD+Abduction (without any consistency checking). From the perspective of the user issuing a naive query, the intensional answer can also be interpreted as the corresponding mediated answer.

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

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

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:

88
\((\exists Y_0, Y_1, Y_2 (r_1(Y_0, Y_1, Y_2), Y_2 = '\text{JPY}' ), \{ N/Y_0, F_1/1\,000, F_2/1 \}) \)

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

### 5.5 Illustrative Example

In this section, we provide an example illustrative of the computation involved inquery 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 \( c_1 \):

\[
\text{Q3: SFECT } r_1.\text{cname}, r_1.\text{revenue}.\text{scaleFactor IN } c1
\]

\[
\text{FROM r1;}
\]

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

\[
\text{CQ3: } \leftarrow \text{answer}(N, F).
\]

\[
\text{answer}(N, F) \leftarrow r_1'(N', R', -), N'[\text{value}(c_1)\rightarrow N], R'[\text{scaleFactor}(c_1)\rightarrow F],
\]

\[
F'[\text{value}(c_1)\rightarrow F].
\]

Figure 5-2 shows one possible refutation of this query using the SLD-Abductionalgorithm described earlier. For better clarity, the refutation is shown using \( \text{IN}\)
clauses rather than Datalog. The clauses used for resolving the goal clauses are thoseshown earlier in Figure 4-1, 4-2 and 4-3.

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

- The refutation begins with the query as given, with \( \Delta \) initialized to the emptyset.

- At step (3), the literal \( r_1(N,-,-) \) cannot be further resolved. Since \( r_1 \) is anextensional predicate (and hence abducible), it is removed from the goal clauseand its Skolemized form, \( r_1(sk_0, sk_1, sk_2) \), is added to \( \Delta \).
Figure 5-2: One possible refutation for query CQ3. Method and functor names are abbreviated where possible (e.g., cr = currency). The resolution step labeled \( \star \) is where a literal is abduced. The abductive answer corresponding to this refutation is given by \( \Delta_4 \), and the intensional answer by \( (\Delta_4, \{N/\text{sk}_0, F/1\}) \).
• At step (6), the literal \( \text{scaleFactor}(c_1, r_{1\#revenue}(sk_0))[\text{value}(c_1) \rightarrow F] \) 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 4-3). This eventually leads to the expansion of the goal clause as shown in step (10).

• In step (12), the extensional relation is referenced again. In the absence of other information, we are not allowed to assume that it is the same “fact” which has been abduced: i.e., we will need to add a new Skolemized fact, \( r_1(sk_3, sk_4, sk_5) \) to \( \Delta \).

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

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

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

\[
\text{SELECT r1.cname, 1 FROM r1 WHERE r1.currency <> 'JPY';}
\]

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 un-intuitive, 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.

5.6 Discussion

It should be pointed that the computation procedure we have described constitutes a "naive" evaluation method. Our intention is not to propose the SLD+Abduction procedure as the most appropriate algorithm; on the other hand, our claim is that the abductive framework provides a useful way of thinking about query mediation and intensional answers which is grounded in a firm logical foundation. The choice of a particular computation procedure for finding the abductive answers corresponding to
a given query, though important, is incidental to the theory which we are proposing. In particular, we note that in order for the SLD+Abduction computation procedure to terminate, the corresponding search space must be finite. When a left-to-right selection rule [Lloyd, 1987] is adopted, this effectively precludes any rules which are left-recursive. A vast literature concerning how some of these problems can be overcomed exists in the logic-programming literature, a detailed discussion of which is unfortunately beyond the means of this Thesis.

Another question which one might ask is the completeness of answers generated via abductive reasoning. It can be easily shown that if all antecedents of a given observation are known, then the disjunction of all abducted answers generated is both sound and complete. In our case, this can be understood as a commitment to a “closed-world” assumption: i.e., if we are able to circumscribe the set of data sources to which a query is posed such that any answer which cannot be derived is considered to be false, then the soundness and completeness results apply.

As we have mentioned earlier in Section 5.1, abductive inferences is in general NP-hard. The computational complexity is a function of the number of context rules which are relevant to a given query, and is a function of the level of nesting of modifiers, the amount of conflicts between the sites that are mediated, and the level of sophistication of the contexts involved. From a practical standpoint, there are a number of strategies which one could take in reducing the overhead of producing a mediated query. As mentioned before, one possibility is allow commonly-encountered queries to be compiled into a view definition which can then be stored, thus avoiding the overhead of having to identify the mediated query each time a user query is submitted. Notice also that not all the context axioms need to be present at all times: in fact, the number of contexts referenced never exceeds the number of sources accessed. By loading only those context axioms which are relevant to a query, we can reduce the number of clauses that resolution will be attempted on. In this sense, the number of context axioms may remain relatively small even though the number of sources and receivers which are being integrated is exponentially large.

Finally, researchers from the deductive database community may find the SLD+
Abduction procedure having much in common with evaluation methods in early deductive database prototypes based on a loose-coupling between relational databases and a Prolog system. For example, a similar procedure was used in EDUCE for collecting a maximal set of references to extensional relations [Ceri et al., 1990] which are subsequently submitted as an SQL query to the underlying relational database. Although the underlying computational methods are similar, there are at least two important differences. First, the abductive framework presents a logical basis for reasoning about an incomplete theory in which extensional facts are abduced to constitute explanations; in a loosely-coupled Prolog-relational system, retrieval of factual statements from the underlying database is an extra-logical step and is viewed as an implementation mechanism that is irrelevant to the underlying deduction-based inferences. This difference in turn lead to a second distinction: unlike the abductive framework, there is little or no systematic attempts at consistency checking. As we have observed above, the ability to reason about abduced answers which are inconsistent with known integrity constraints are important for both conceptual and operational reasons.
Chapter 6

Data Modeling in COIN

In this chapter, we present an in-depth case study of an actual integration scenario using the integration strategy proposed in this Thesis. Our primary goal is to illustrate the expressive power of the Context Interchange framework by demonstrating how various types of conflicts encountered in a real-world scenario can be dealt with using the concepts introduced. A secondary goal is to offer some guidelines for domain and context modeling: whenever appropriate, we present alternative representations and discuss their tradeoff. While a detailed investigation into modeling methodologies is beyond the scope of this Thesis, we believe that the insights presented via this case study will be helpful for providing a jump-start in this direction.

6.1 The Integration Scenario

The scenario for our case study is a variation of the motivational example, but is grounded in actual data sources. We begin with the three data sources shown in Figure 6-1. Both DISCLOSURE and WORLDSCOPE are Oracle databases, each consisting of a single relation (discaf and worldaf respectively) that describe the financial performance of companies. The third data source, OLSEN, is a web-site which furnishes the exchange rate of any two currencies for a given date. Access to this web-site is mediated by a “wrapper” (see Chapter 7) which, for all intents and purposes, allows us to treat the web-site as if it were another relational database. A subset of the data
for these databases are shown in Figure 6-1.

<table>
<thead>
<tr>
<th>DISCLOSURE DATABASE</th>
<th>discAF( companyName, fiscalDate, netIncome, netSales, locIncorp )</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Daimler Benz Corp'</td>
<td>12/31/93 615 000 000 97 737 000 000 'GERMANY'</td>
</tr>
<tr>
<td>'Nippon Telegraph &amp; Telephone Corp'</td>
<td>03/31/94 83 396 000 5 907 988 000 'JAPAN'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WORLDSCOPE DATABASE</th>
<th>worldAF( companyName, latestFinDate, netIncome, Sales )</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Daimler-Benz AG'</td>
<td>12/31/93 346 577 56 268 .68</td>
</tr>
<tr>
<td>'Nippon Telegraph &amp; Telephone Corp'</td>
<td>03/31/94 486 515 64 860 598</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OLSEN</th>
<th>exchange( sourceCurr, targetCurr, date, rate )</th>
</tr>
</thead>
<tbody>
<tr>
<td>'German Mark'</td>
<td>'US Dollar'</td>
</tr>
<tr>
<td>'German Mark'</td>
<td>'US Dollar'</td>
</tr>
<tr>
<td>'Japanese Yen'</td>
<td>'US Dollar'</td>
</tr>
<tr>
<td>'Japanese Yen'</td>
<td>'US Dollar'</td>
</tr>
</tbody>
</table>

Figure 6-1: Integration scenario.

For the present discussion, we will limit the semantic conflicts among these data sources to those described below. (Further variations will be introduced later to illustrate other features of interest to us.)

- **Conflict over company names.** Contrary to what one might expect, the names of companies in the DISCLOSURE and WORLDSCOPE databases are reported differently. For example, both 'Daimler Benz Corp' in DISCLOSURE and 'Daimler-Benz AG' in WORLDSCOPE refer to the same entity in the real-world. Unlike conflicts such as "scale-factor", these differences here are idiosyncratic in that there often are no systematic rules for mapping from one representation to another although there are often multiple widely-adopted standards for company names (e.g., Reuters, Dun & Bradstreet).

- **Conflict over scale-factors and currencies.** Similar to the earlier motivational
example, the figures for netIncome and netSales in both DISCLOSURE and WORLDSCOPE differ in both currencies and scale-factors. In the case of DISCLOSURE, all “company financials” are reported using a scale-factor of 1, and in the “official currency” of the country in which the company is “incorporated”. For example, both income and sales figure for 'Daimler Benz Corp' are reported in 'German Mark' since the company is incorporated in 'Germany'. On the other hand, all “company financials” in WORLDSCOPE are reported using a scale-factor of 1000 and using 'US Dollar'.

- Conflict over date formats. Both DISCLOSURE and WORLDSCOPE report dates in the format mm/dd/yy, which is different from those in OLSEN, where the format used is dd/mm/yy.

As we will show later in this chapter, these conflicts can be resolved using various conversion functions which are representative of the possible types of conversions supported by the Context Interchange strategy.

It is interesting to point out that data present in a source may in fact be derived from one or more sources. For example, in the case of WORLDSCOPE, some “company financials” are obtained from other “up-stream” sources (e.g., DISCLOSURE) and converted to US currency. This process may introduce “noise” or other assumptions into the data which are not an inherent part of the “base data”. For example, currency conversion necessitates the use of some exchange rate, the value of which depends not only on the currencies involved, but also the reference time-point (since exchange rates varies from day to day). For example, the exchange rate chosen may in fact be that corresponding to the latest financial-reporting date (i.e., '12/31/93' in the case of the tuple corresponding to 'Daimler-Benz AG' shown in Figure 6-1), or it could have been the exchange rate “as of” today, or of any arbitrary date (say, “as of” the day on which the data is loaded into the database). In general, this information constitutes a piece of meta-data which needs to be made explicit in the corresponding context if variations in exchange rates are important for the problem domain we are interested in.
6.2 Domain Modeling

Figure 6-2 depicts one feasible domain model for the integration exercise at hand. Like virtually all design tasks, domain modeling is a creative activity and is capable of admitting multiple solutions. (An analogy is drawn here with conceptual data modeling in the ER tradition: with a decade or more active research, there remains little consensus on what constitutes a good design\(^1\)).

![Domain Model Diagram]

Figure 6-2: A domain model for the integration scenario. (Note that companyName, countryName, dateFormat, etc are subtypes of the type semanticString. These subtype relations, along with the type semanticString are not shown in the diagram for clarity reasons.)

This domain model shown above is similar to the one presented earlier in Chapter 3. We elaborate below some of the differences which are of interest to us:

- As before, companyFinancials is introduced as a subtype of moneyAmt. We add a new structural attribute, periodEnding so that we may meaningfully refer to

\(^1\)See for instance, [Goh and Ling, 1992], for one such discussion.
companyFinancials pertaining to different financial reporting periods (e.g., for different financial years).

- In addition to modifiers currency and scaleFactor, the semantic-type companyFinancials has a new modifier labeled derivedAsOf, which returns an instance of the type date. This modifier captures the fact that the moneyAmt reported may have been derived and thus provide a reference date on which this conversion took place.

- To capture variations in the representation of company names and dates, we associate modifiers nameFmt and dateFmt with semantic-types companyName and date respectively.

### 6.3 Mapping Sources to the Domain Model

The mapping of data and data-relationships in sources to the domain model is accomplished via the elevation axioms. As formally described in the Context Interchange framework (see Section 4.5), this step consists of three distinct operations:

1. defining a “virtual” semantic-relation corresponding to each extensional relation;

2. assigning to each semantic-object, its value in the context of the source, as derived from the extensional relation; and

3. mapping semantic-objects in the semantic-relation to types in the domain model while making explicit any implicit links represented by the semantic-relation.

We exemplify these operations for each of the sources below.

**Definition of Semantic-Relations**

For each extensional relation, a semantic-relation is defined on the semantic-objects corresponding to the cells of the earlier. Thus, we have
\[\text{discaf}'(f_{\text{discaf}\#\text{cn}}(N, D), f_{\text{discaf}\#\text{dt}}(N, D), f_{\text{discaf}\#\text{inc}}(N, D), f_{\text{discaf}\#\text{ns}}(N, D), f_{\text{discaf}\#\text{loc}}(N, D)) \leftarrow \text{discaf}(N, D, \_, \_, \_).
\]
\[\text{worldaf}'(f_{\text{worldaf}\#\text{cn}}(N, D), \ldots) \leftarrow \text{worldaf}(N, D, \ldots).
\]
\[\text{olsen}'(f_{\text{olsen}\#\text{fc}}(F, T, D), \ldots) \leftarrow \text{olsen}(F, T, D, \_).
\]

As we have pointed out before, the choice of a Skolem function which is unique to each relation-attribute pair, coupled with the use of the "key" of each relation, provides the mechanism for generating the Skolem-objects which are uniquely associated with each cell of the extensional relation. In other words, any two source data elements will always be mapped to distinct objects (having distinct oids, as represented by the Skolem term) even though they may have the same value. This is true whether they are drawn from two distinct relations (in separate databases) or from the same relation. For example, the value 'Daimler Benz Corp' may occur several times in the discaf relation (once for each financial year being reported, say '12/31/93' and '12/31/94'), but each occurrence of which will result in a different oid-term (in this case, \(f_{\text{discaf}\#\text{cn}}('\text{Daimler Benz Corp}', '12/31/93')\) and \(f_{\text{discaf}\#\text{cn}}('\text{Daimler Benz Corp}', '12/31/94')\)). Similarly, two company names that appear separately in DISCLOSURE and WORLDSCOPE may be identical, but will be represented as distinct semantic-objects since the Skolem functions will be different.

**Assignment of Values to Semantic-Objects**

As we have pointed out earlier, the semantic-objects introduced in the semantic-relations may have distinct values in different contexts. By virtue of the way oid-terms are created, each semantic-object is indigenous to some semantic-relation (and consequently, exactly one data source). The value of a semantic-object in the context of the source (of which the semantic-object is indigenous to) is given in the elevation axioms. For instance, we may define the following axioms which provide the value assignments to the semantic-objects referenced in the semantic-relation corresponding to the discaf relation:

\[f_{\text{discaf}\#\text{cn}}(N, D)[\text{value}(C) \rightarrow N] \leftarrow \mu(\text{discaf}, C), \text{discaf}(N, D, \_, \_, \_).
\]
\[ f_{\text{discaf\#dt}}(N, D)[\text{value}(C) \rightarrow D] \leftarrow \mu(\text{discaf}, C), \text{discaf}(N, D, \_ , \_ , \_ ) . \]
\[ f_{\text{discaf\#inc}}(N, D)[\text{value}(C) \rightarrow I] \leftarrow \mu(\text{discaf}, C), \text{discaf}(N, D, I, \_ , \_ ) . \]
\[ f_{\text{discaf\#ns}}(N, D)[\text{value}(C) \rightarrow S] \leftarrow \mu(\text{discaf}, C), \text{discaf}(N, D, \_ , S, \_ ) . \]
\[ f_{\text{discaf\#loc}}(N, D)[\text{value}(C) \rightarrow L] \leftarrow \mu(\text{discaf}, C), \text{discaf}(N, D, \_ , \_ , L) . \]

In the above axioms, the predicate \( \mu \) identifies the context which is bound to the the \textsc{disclosure} database, of which relation \text{discaf} is a part of. This provides an added level of indirection which allows values in a source to be associated with the corresponding semantic-objects without determining a priori which context are bound the source.

**Mapping to the Domain Model**

The final component of the elevation axioms allows semantic-objects and implicit relationships embodied in a semantic-relation to be mapped to types and explicit attribute links in the domain model. In the case of \text{discaf}, we may thus write

\[ N' : \text{companyName} \leftarrow \text{discaf}'(N', \_ , \_ , \_ , \_ ) . \]
\[ D' : \text{date} \leftarrow \text{discaf}'(\_ , D', \_ , \_ , \_ ) . \]
\[ F' : \text{companyFinancials} \leftarrow \text{discaf}'(\_ , \_ , F', \_ , \_ ) . \]
\[ F' : \text{companyFinancials} \leftarrow \text{discaf}'(\_ , \_ , F', \_ , \_ ) . \]

\[ \ldots \]
\[ N' : \text{companyName} \vdash N'[\text{countryIncorp} \rightarrow L'] \leftarrow \text{discaf}'(N', \_ , \_ , \_ , L') . \]
\[ F' : \text{companyFinancials} \vdash R'[^{\text{fyEnding}} \rightarrow D'] \leftarrow \text{discaf}'(\_ , D', F', \_ , \_ ) . \]
\[ F' : \text{companyFinancials} \vdash R'^{\text{fyEnding}} \rightarrow D' \leftarrow \text{discaf}'(\_ , D', \_ , F', \_ ) . \]
\[ F' : \text{companyFinancials} \vdash R'^{\text{ofCompany}} \rightarrow N' \leftarrow \text{discaf}'(N', \_ , F', \_ , \_ ) . \]
\[ F' : \text{companyFinancials} \vdash R'^{\text{ofCompany}} \rightarrow N' \leftarrow \text{discaf}'(N', \_ , F', \_ , \_ ) . \]

\[ \ldots \]

The mapping of semantic-objects to semantic-types in the domain model provides the means for deciphering the semantics of data elements in a source or receiver. The instantiation of the attribute links is an added feature which allows us to make references to relationships among semantic-objects without explicit reference to any
particular schema. This step can in fact be understood as elucidating the implicit relationships embodied in a database schema, and allow context descriptions to be formulated independently of the schemas corresponding to the underlying sources.

6.4 Context Definition

Consistent with our earlier formulation of the Context Interchange framework, we may define distinct collections of context axioms which are used as the basis for describing the semantics of individual data elements within sources and receivers. The context axioms can be conveniently partitioned into the set of modifier assignments and the set of conversion function definitions. We deal with each of these in turn below.

For the example at hand, we assume the existence of three contexts: context $c_{DS}$ corresponding to the DISCLOSURE database, context $c_{WS}$ corresponding to the WORLDSCOPE database, and $c_{OS}$ corresponding to the OLSEN web-site.

Modifier Assignments

The assignment of modifier values are straight-forward for most part. In the case of context $c_{DS}$, we may write:

$$D' : date, f_{dateFmt(c_{OS})}(D') : dateFormat \leftarrow$$

$$f_{dateFmt(c_{OS})}(D')[value(c_{DS}) \rightarrow 'mm/dd/yy'].$$

$$N' : companyName, f_{nameFmt(c_{OS})}(N') : nameFmt \leftarrow$$

$$f_{nameFmt(c_{OS})}(N')[value(c_{DS}) \rightarrow 'ds_names'].$$

$$F' : companyFinancials, f_{scaleFactor(c_{OS})}(F') : semanticNumber \leftarrow$$

$$f_{scaleFactor(c_{OS})}(F')[value(c_{DS}) \rightarrow 1].$$

$$F' : moneyAmt, f_{currency(c_{OS})}(F') : currencyType \leftarrow$$

$$f_{currency(c_{OS})}(F')[value(c_{DS}) \rightarrow 'German Mark']\leftarrow$$

$$F'[ofCompany \rightarrow N_1'], N_2' : companyName[countryIncorp \rightarrow T'], N_1' \equiv N_2' \equiv T'[value(c_{DS}) \rightarrow 'Germany'].$$
\[ f_{\text{currency}}(c_{\text{ds}})(F')[\text{value}(c_{\text{ds}}) \rightarrow '\text{Japanese Yen'}] \leftarrow \]
\[ F'[\text{ofCompany} \rightarrow N'_1], N'_2 : \text{companyName}[\text{countryIncorp} \rightarrow T'], N'_1 \approx N'_2 \]
\[ T'[\text{value}(c_{\text{ds}}) \rightarrow '\text{Japan'}]. \]

\[ f_{\text{currency}}(c_{\text{ds}})(F')[\text{value}(c_{\text{ds}}) \rightarrow '\text{US Dollar'}] \leftarrow \]
\[ F'[\text{ofCompany} \rightarrow N'_1], N'_2 : \text{companyName}[\text{countryIncorp} \rightarrow T'], N'_1 \approx N'_2 \]
\[ T'[\text{value}(c_{\text{ds}}) \rightarrow '\text{United States'}]. \]

The first three axioms above trivially asserts that the date format, company-name format, and scale-factor are respectively (and unconditionally) 'mm/dd/yy', 'disclosure_names' and 1 in the context \( c_{\text{ds}} \). The remaining axioms define the currency in use as being contingent on the country-of-incorporation corresponding to the company referenced.

In the axioms concerning currency-assignments, notice that we require only semantic-equivalence between instances of \text{companyName} (i.e., \( N'_1 \approx N'_2 \)), which is a weaker condition than oid-equivalence (i.e., \( N'_1 = N'_2 \)). This is significant because it does not impose, a priori, the constraint that the location-of-incorporation of a company be available in the same relation and more importantly, provide for mediated access to other sources which may provide information that is useful for defining the current context. We illustrate this remark by considering an alteration to the schemas for \text{DISCLOSURE} and \text{WORLDSCOPE} databases as shown in Figure 6-3, where the only change is for location-of-incorporation to be reported in \text{WORLDSCOPE} as opposed to \text{DISCLOSURE}.
\[
\text{Figure 6-3: Alteration to the DISCLOSURE and WORLDSCOPE schemas resulting in location-of-incorporation being reported in the WORLDSCOPE database instead.}
\]

Under this scenario, the elevation axioms allow us to infer the following "facts" (among others):

\[
\begin{align*}
\text{\( f_{\text{discaf}\#n}(\text{"Daimler Benz Corp",}'12/31/93') : } & \text{netIncome} \\
& \text{[ofCompany} \rightarrow f_{\text{discaf}\#cn}(\text{"Daimler Benz Corp",}'12/31/93')).] \\
\text{\( f_{\text{worldaf}\#cn}(\text{"Daimler-Benz AG",}'12/31/93') : } & \text{companyName} \\
& \text{[countryIncorp} \rightarrow f_{\text{worldaf}\#loc}(\text{"Daimler-Benz AG",}'12/31/93')).]
\end{align*}
\]

Despite these changes, the context axioms described earlier are still valid. In particular, although the semantic-objects represented by oid-terms \( f_{\text{discaf}\#cn}(\text{"Daimler Benz Corp", }'12/31/93') \) and \( f_{\text{worldaf}\#cn}(\text{"Daimler-Benz AG", }'12/31/93') \) (corresponding to one possible binding of \( N_1 \) and \( N_2 \) respectively) are not oid-equivalent, they are nevertheless semantically-equivalent with respect to context \( c_{\text{ns}} \). This is because both objects have the same value ("Daimler Benz Corp") in context \( c_{\text{ns}} \) despite the fact that they are represented differently in their respective sources.

It is also useful to point out that alternative formulation of the context rules are possible. For example, instead of having one axiom to each currency, we may have only one axiom which references yet another data source for the official currency of
each country. For example, suppose we have a source with the following schema\footnote{Not surprisingly, this information is already available on the Web in a number of sites, one of which is the CIA Factbook. In the current prototype, this web-site can be accessed through a \textit{``wrapper''}, which provides a relational interface that accepts queries formulated in SQL.}:

\[
\text{factbook(c\text{ountry}, c\text{urrency})}
\]

in which each tuple describes the official currency in use in a country.

Suppose this source is mapped to the domain model in a manner similar to our earlier description for the other sources. We can replace \textit{all} of the context axioms for currency-assignment with the following:

\[
F' : \text{companyFinancials}, f_{\text{currency(c\text{os})}}(F') : \text{currencyType} \vdash \\
\quad f_{\text{currency(c\text{os})}}(F')[\text{value(c\text{os})}\rightarrow Y] \leftarrow \\
\quad F'[\text{ofCompany}\rightarrow N_1'], N_2' : \text{companyName}[\text{countryIncorp}\rightarrow T_1'], N_1' \overset{\text{cos}}{=} N_2' \\
\quad T_2' : \text{countryName}[\text{officialCurrency}\rightarrow Y'], T_1' \overset{\text{cos}}{=} T_2', Y'[\text{value(c\text{os})}\rightarrow Y].
\]

The tradeoff between the two formulations can be understood at two levels. First, the latter form is more compact and thus less cumbersome; this is advantageous from a specification point of view. Second, the abductive procedure is more efficient with the second representation since there is now only one clause which can resolve requests for currency of \textit{companyFinancials}. Since this currency information is present in an extensional data source, the mediated query will need to perform a join on this source to obtain the actual currency value corresponding to each item. This can be accomplished with a single query, in contrasts with the earlier context formulation in which each currency assignment results in a different subquery. The cost of executing the mediated queries corresponding to the two alternative formulations are unfortunately unclear without more information on the distribution of values in each data source (e.g., are there five hundred different countries or just three) and also the exact form of the query.

One possible complication here is that it may no longer be possible to identify certain spurious conflicts a priori during context mediation. For example, if a query had asked for companies that reports only in “Japanese Yen”, we would not have
known that these are exactly those companies which are incorporated in "Japan", and would have rewritten the mediated query in a manner that requires a join-operation with relation factbook. There are however different strategies which one might be able to take in implementing the abduction engine to account for these types of anomalies. In the above example, if it is true that every country uses a different currency, we could use the underlying functional dependency currency → country for inferring the fact that there is exactly one country name which corresponds to any given currency, and thus initiate a database access returning one tuple from within the Context Mediator. This would have allowed us to eliminate the more expensive (and less intuitive) join-operation in the corresponding mediated query.

Finally, the value corresponding to modifier derivedAsOf can be similarly defined depending on the semantics of the data present in the sources. For example, if it is the case that financial figures in WORLDSCOPE are based on the currency exchange rates of "today", we can represent this with the axiom:

\[
F' : companyFinancials, f_{\text{derivedAsOf}}(c_{\text{ws}})(F') : date \leftarrow \\
f_{\text{derivedAsOf}}(c_{\text{ws}})(F')[\text{value}(c_{\text{ws}}) \rightarrow D] \leftarrow \text{today}(D).
\]

where the predicate today is declared as an external function which can be invoked to return the date today. Alternatively, if the conversion is always based on the date corresponding to the end of each financial-year, this can be represented as

\[
F' : companyFinancials, f_{\text{derivedAsOf}}(c_{\text{ws}})(F') : date \leftarrow \\
f_{\text{derivedAsOf}}(c_{\text{ws}})(F')[\text{value}(c_{\text{ws}}) \rightarrow D] \leftarrow \\
F'[\text{fyending} \rightarrow D'], D'[\text{value}(c_{\text{ws}}) \rightarrow D].
\]

It should be clear from the preceding discussion that the COIN language is sufficiently expressive for encoding any general rule.

**Conversion Function Definitions**

Conversion functions for reconciling conflicting data representations and interpretations can be realized in a number of ways. We present below a loose typology through
representative examples which are relevant to the integration scenario presented earlier.

1. Arithmetic functions.

The simplest type of conversion functions is probably arithmetic operations on data. An example of this is scale-factor conversion which was encountered earlier. Hence, to account for differences in scale-factor between WORLDSCope and DISCLOSURE, we could write the following conversion function:

\[ M' : \text{moneyAmt} \mapsto M'[\text{cvt}(C_2)@\text{scaleFactor}, C_1, U \to V] \leftarrow \]
\[ f_{\text{scaleFactor}(C_1)}(M')[[\text{value}(C_2) \to F_1], f_{\text{scaleFactor}(C_2)}(M')[[\text{value}(C_2) \to F_2], \]
\[ V = U \times F_1 / F_2. \]

Notice that the above definition does not make any assumption of the scale-factor of \( C_2 \) (but request for it explicitly in the body of the clause). This is important for at least two reasons. First, it allows changes to be made to the context axiom for scale-factor without having an impact on the conversion function. Second, this conversion function may be "instantiated" in different contexts which may have different scale-factor assignments. (This observation applies to all conversion functions which are discussed below.)

2. Table lookup.

One of the most effective mechanisms for dealing with idiosyncratic conflicts is to resolve the differences via a one-to-one mapping which can be represented in an extensional table. In our example, this is the strategy adopted for resolving conflicts on \text{company\_name}. Thus, we may create a table, referred to by \text{namemap}, whose schema might be given by:

\[ \text{namemap( ds\_names, ws\_names )} \]

which identifies the matching names in the DISCLOSURE and WORLDSCope databases. For convenience, we introduce an intensionally-defined predicate,
lookup/4\(^3\), given by

\[
\begin{align*}
\text{lookup}(\text{'ds\_names'}, \text{'ws\_names'}, X, Y) & \leftarrow \text{namemap}(X, Y). \\
\text{lookup}(\text{'ws\_names'}, \text{'ds\_names'}, X, Y) & \leftarrow \text{namemap}(Y, X)
\end{align*}
\]

This allows us to define the conversion function for companyName as shown below:

\[
N' : \text{companyName} \leftarrow N'[\text{cvt}(C_2)@\text{cnameFormat}, C_1, U \rightarrow V] \leftarrow \\
\text{f}_{\text{cnameFormat}(C_1)}[\text{value}(C_2) \rightarrow F_1], \text{f}_{\text{cnameFormat}(C_2)}[\text{value}(C_2) \rightarrow F_2], \\
\text{lookup}(F_1, F_2, U, V).
\]

It should be obvious that the namemap table can be extended to allow for arbitrary number of columns by modifying the definition of lookup to query the schema for the namemap relation. In which case, changes to the namemap relation would not require changes to the conversion functions (or the intensional rule lookup). Notice also that namemap need not be a materialized relation; thus, we may have base tables which provide for pairwise mapping between different naming conventions, with namemap defined as a view. The choice of one for the other is orthogonal to the inferences needed to support context mediation.

3. External Functions.

Finally, the use of external functions provide a "catch-all" hatch for dealing with arbitrarily complex conversions. In the example at hand, conversions between different date formats require somewhat more complex string-manipulations. Suppose this transformation is implemented in some external procedure

\[
\text{cvtDate}(Format_1, Format_2, Representation_1, Representation_2)
\]

\(^3\)Following Prolog's convention, the token p/n is used to denote a predicate with name p and arity 4.
we can define the conversion function within the COIN framework as shown below:

\[
D': \text{date} \leftarrow D'[\text{cvt}(C_2)\circ\text{dateFmt}, C_1, U\rightarrow V] \leftarrow
\]

\[
f_{\text{dateFmt}(C_1)}[\text{value}(C_2)\rightarrow F_1], f_{\text{dateFmt}(C_2)}[\text{value}(C_2)\rightarrow F_2],
\]

\[
\text{cvtDate}(F_1, F_2, U, V).
\]

(For efficiency reasons, we probably do not want to invoke an external function if the two formats happen to be identical. Thus, the above definition can be written as two separate clauses: in the first case, if \(F_1 = F_2\), we simply return \(V = U\); otherwise \(F_1 \neq F_2\), we then require the external function to be invoked. This distinction however is irrelevant from a logical specification point of view.)

Where it is feasible, we generally prefer not to encapsulate transformation in external functions since this creates a black-box which is problematic from a query optimization standpoint. This suggests that if efficiency is a concern, we can impose structure on the data that allow us to operate on them at a finer level of granularity. This is the case with the domain of real numbers, in which the set of arithmetic operations constitute primitive operators that can be assembled to provide complex transformations without making the data transformation opaque.

6.5 Discussion

As we have demonstrated in the preceding sections, a great deal of flexibility exists for modeling the semantics of data in widely disparate contexts. A strategic decision underlying the Context Interchange approach is the commitment to modeling semantics of data at the level of individual data elements: while this has presented opportunities for encapsulating the semantics of data in the corresponding semantic types, it presents difficulty when there is a boundary mismatch between data elements across disparate sources. For example, two data sources may report date-information somewhat differently as shown below:
\[ r1(\text{day, month, year, ...}) \]
\[ r2(\text{date, ...}) \]

There are several alternatives for dealing with such mismatches. We describe two of these strategies below with a brief discussion of their relative merits.

**Alternative 1**

If we allow access to data sources to be mediated by views, the above mismatch can be solved by defining a view on \( r1 \) which combines the three attributes (day, month, and year) into a singleton-attribute:

\[ \text{view1(T, ...)} \leftarrow r1(D, M, Y, ...), \text{compose\_date}(D, M, Y, T). \]

Rather than mapping \( r1 \) directly to the domain model, we might define the mapping on \( \text{view1} \) instead. Although the anomaly is no longer present, this approach may be problematic since this may be easily abused to create a multitude of opaque transformations in the view definition which impedes our ability to reason about semantics of data native to each system.

**Alternative 2**

A second alternative to the above anomaly is to enrich the domain model so that it can explicitly account for possible derivations of a value corresponding to a semantic-object, by drawing upon values of others. In the example at hand, we may associate a conversion function with the semantic-type *date* which is not unlike \( \text{compose\_date} \). In this instance, this can be understood as "pushing" the view definition into the domain model. The advantage of this over the first alternative is that it allows common knowledge (in this case, how to derive a date value from values of day, month and year) to be captured as part of a domain model. From a pragmatic point of view, this creates a problem because we would end up with axioms in the domain model which are mutually exclusive: for example, we may have a clause which defines how "year" might be derived from "date", and yet another clause which allows "date"
to be derived from "day", "month" and "year". This recursion creates problems for the SLD+Abduction algorithm which must now be extended to avoid being trapped on a branch of the SLD-tree which is infinite.
Chapter 7

The Context Interchange Prototype

The goal of the Context Interchange Prototype is to provide a demonstration of the feasibility and features of the Context Interchange strategy. In commensuration with the goals of this Thesis, we are particularly interested in a concrete implementation of the Context Mediator. In Section 7.1, we present the architecture of this Prototype and a brief description of its components. Section 7.2 contains an in-depth discussion of the implementation of the Context Mediator, which is responsible for transforming a user query to a mediated query. The last section concludes this chapter by portraying how a user might make use of the system to gain “mediated access” to disparate information sources on the World Wide Web.

7.1 The Context Interchange Prototype: Overview

Figure 7-1 shows the architecture of the Prototype which is being implemented, which consists of three distinct groups of processes.

- *Client Processes* provide the interaction with receivers and route all database requests to the Context Mediator. An example of a client process is the *multi-database browser* [Jakobisiak, 1996], which provides a point-and-click interface
Figure 7-1: Architectural overview of the Context Interchange Prototype.
for formulating queries to multiple sources, and for displaying the answers obtained. More generally, any application program which issues queries to one or more sources can be considered a client process. For example, Microsoft Excel has the capability of issuing an SQL-query encapsulated as an ODBC-request. In the current implementation, this request is intercepted by a custom ODBC driver, which redirects the request to the Context Mediator.

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

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

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

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

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

The Context Mediator is implemented in ECLiPSe\textsuperscript{1}, which is an efficient and robust Prolog implementation distributed by the ECRC. In actuality, the Context Mediator consists of four distinct components which are loosely-connected to one another as shown in Figure 7-2. At the heart of the Context Mediator is the Abduction Engine which implements the extended SLD+Abduction algorithm as was described in Chapter 5. Since computation of the abductive answer is performed within a Horn-clause (HC) framework, we need to translate both the user-query as well as \texttt{coin} clauses to HC (or equivalently, statements in Datalog\textsuperscript{neg}), and on obtaining the answer, perform the reverse translation to SQL. Both the SQL-to-HC and HC-to-SQL translation are relatively straight-forward since both of these languages shares a common grounding in predicate calculus. In the current implementation, the \texttt{coin} clauses precompiled and stored within the system in the form of Horn clauses. The translation of an SQL-query to HC and vice versa are performed dynamically whenever a query is submitted.

The Abduction Engine

The Abduction Engine takes the form of a \textit{meta-interpreter} [Sterling and Shapiro, 1994, Chapter 17], the skeleton of which is shown in Listing 7.2.1.

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

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

\textsuperscript{1}ECLiPSe: The ECRC Constraint Logic Parallel System. More information can be obtained at http://www.ecrc.de/eclipse/.
Listing 7.2.1 Skeleton of the meta-interpreter implementing the Abduction Engine.

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

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

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

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

\[
\text{abducible}(L) :- \text{prolog}((\text{functor}(L,\text{Functor},\text{Arity})), \\
\text{abducible_predicate}(\text{Functor},\text{Arity}).
\]

\[
\text{abducible_predicate}(\text{is},2).
\]

\[
\text{abducible Predicate}(\text{r1},3).
\]

\[
\text{abducible Predicate}(\text{r2},2).
\]

All extensional database predicates are declared to be abducible predicates in the Enriched Schema. The declaration of \text{is}/2 as abducible allows arithmetic expressions in general to be part of the abductive answer. (Following Prolog's convention, we refer to a predicate \text{p} with arity \text{n} by the token \text{p/n}. Predicates with the same name (\text{p}) but different arity are treated as distinct; hence, the declaration of which predicates are abducible requires two arguments as shown above.)

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

The Abductive Query

The abductive procedure is initiated by invoking \text{abductively_solve} as follows:

\[
:- \text{abductively_solve}(\text{Goal},[],\text{Ans},[],\text{Constr}).
\]

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

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

\[
\text{answer}(C,R) :- r_1(p(E_1,X,\_)), r_2(p(E_2,Y), \\
\quad \text{sem\_op}(=,c_2,E_1,E_2), \text{sem\_op}(>,c_2,X,Y), \\
\quad \text{value}(E_1,c_2,C), \text{value}(X,c_2,R).
\]

The abductive answer is obtained by asking the query

\[
:- \text{abductively\_prove(answer}(C,R),[],\text{Ans},[],\text{Constr}).
\]

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

\[
C = X, \ R = Y, \\
\text{Ans} = [r_1(X,Y,U), r_2(X,Z), Y>Z], \text{Constr} = [U='USD']
\]

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

\[
C = X, \ R = Y_1, \\
\text{Ans} = [r_1(X,Y,U), r_2(X,Z), r_3(U,V,T), Y_0 \text{ is } Y*1000, Y_1 \text{ is } Y_0*T, Y_1>Z], \\
\text{Constr} = [U='JPY', V='USD']
\]

and

\[
C = X, \ R = Y_1, \\
\text{Ans} = [r_1(X,Y,U), r_2(X,Z), r_3(U,V,T), Y_1 \text{ is } Y*T, Y_1>Z], \\
\text{Constr} = [U<>'JPY', U<>'USD', V='USD']
\]

These abductive answers transform directly to the mediated query MQ1 shown earlier in Chapter 3.
7.3 Mediated Data Access: A User’s Perspective

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

The multidatabase browser presents a convenient interface to all the sources which are accessible in the current system. At the present moment, this includes both Oracle databases containing financial information regarding companies as well as web-sites providing stock quotes and other financial information (e.g., currency exchange rates). As illustrated in Figure 7-3, a query can be formulated via the point-and-click interface provided by the multidatabase browser. In this example, the attributes which are “checked” (in the leftmost check-boxes) are those which should appear in the target-list of the query. The selection criteria in the query is represented by constraints on attributes selected by a user. It is obvious that queries formulated using this interface can be easily rewritten into SQL queries in conjunctive form.

The query formulated by a user can be executed with or without being mediated. If the mediation “button” is not checked, the query will be executed “as is” and the Context Mediator is by-passed altogether. If “mediation” is turned on, two other sets of options are available. First, the user must select a context for executing the query. This context is used both for semantic comparisons as well as for reporting the values of data requested in a query. Second, the multidatabase provides an option for examining a trace of the underlying mediation and execution. If the “trace mode” button is checked, the intermediary steps leading to the answer will be displayed. This feature is useful for understanding the behavior of the system and for debugging purposes.
Figure 7-3: Screen-shot of the multidatabase browser.
Chapter 8

Conclusion

In this Thesis, we have presented a formal model for the Context Interchange strategy, characterized by a COIN framework, which is in turn constructed upon a COIN data model. We submit that this formalization is useful for a variety of different reasons.

First, the adoption of a logical framework provides us with a well-founded basis for describing the semantics of data in disparate systems. Thus, instead of relying on arbitrary algorithms for reasoning about data semantics (e.g., the subsumption algorithm [Siegel and Madnick, 1991]), we are able to exploit results in mathematical logic (in particular, logic programming) which are well-understood. At the same time, this connection to logic has allowed us to incorporate a number of new functionalities. For example, instead of merely identifying the potential conflicts which may be present between two systems, we show that a mediated query can be constructed using abductive inferences. Moreover, the same "machinery" can be used for supporting queries on data semantics (knowledge-level queries), for returning intensional answers, and for identifying and pruning spurious conflicts which are guaranteed never to occur.

Second, the COIN framework also teases apart the notion of contexts and schemas. Unlike the semantic-value model [Sciore et al., 1994] where contexts are tightly bound to schemas using annotations in the form of meta-attributes, our formulation requires only a loose-coupling between the two. Specifically, contexts are defined with references to the domain model and are independent of specific schemas whenever feasible. This allows the same context to be shared among distinct sources and receivers, and
presents a novel opportunity for sustaining an infrastructure for support semantic interoperability. This is further facilitated by the "object-oriented" features of the COIN data model, which allows us to take advantage of encapsulation, inheritance, and overriding.

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

From the preceding discussion, it should be clear that the Context Interchange strategy presents many novel features 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. The Context Interchange approach provides a novel middle ground between the two: it allows queries to sources to be mediated in a transparent manner, provides systematic support for elucidating the semantics of data in disparate sources and receivers, and at the same time, does not succumb to the complexities inherent in maintenance of shared schemas.

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

- framing the problem in McCarthy's theory of contexts [McCarthy, 1987] (as in Carnot [Collet et al., 1991], and more recently, [Faquhar et al., 1995]);

- encapsulation [Atkinson et al., 1989] of semantic knowledge in a hierarchy of rich data types which are refined via sub-typing (as in several object-oriented
multidatabase systems, the archetype of which is Pegasus [Ahmed et al., 1991]);

- adoption of a deductive or object-oriented formalism [Kifer et al., 1995, Dobbie and Topor, 1995] (as in the ECRC Multidatabase System [Jonker and Schütz, 1995] and DISCO [Tomasic et al., 1995]);

- provision of value-added services through the use of mediators [Wiederhold, 1992] (as in TSIMMIS [Garcia-Molina et al., 1995]);

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

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

In the Carnot system [Collet et al., 1991], semantic interoperability is accomplished by writing articulation axioms which translate "statements" which are true in individual sources to statements which are meaningful in the Cyc knowledge base [Lenat and Guha, 1989]. A similar approach is adopted in [Faquehar et al., 1995], where it is suggested that domain-specific ontologies [Gruber, 1991], which
may provide additional leverage by allowing the ontologies to be shared and reused, can be used in place of Cyc. While we like the explicit treatment of contexts in these efforts and share their concern for sustaining an infrastructure for data integration, our realization of these differ significantly in that lifting axioms [Guha, 1991] in our case operate at a finer level of granularity: rather than writing axioms which map “statements” present in a data source to a common knowledge base, they are used for describing “properties” of individual “data objects”. This difference accounts largely for the scalability and extensibility of our approach.

The TSIMMIS [Papakonstantinou et al., 1995, Quass et al., 1995] approach developed at Stanford University is developed based on 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.

Finally, we note that some of these research questions have also attracted the interests of researchers from the artificial intelligence (AI) community. A key difference between these two streams of research lies in that nature of “information” which are being exchanged: research arising from the AI community are much more concerned with complex knowledge (e.g., statements in first-order predicate language as opposed

---

1 See also discussion in Section 4.2 for an in-depth discussion of the differences between the two “kinds” of lifting axioms.
to factual statements in databases) and the sharing of resources which may involve different computational capabilities (e.g., specialized inference methods that are present in disparate systems). At the same time, there is much less focus on issues pertaining to efficiency and scalability since the amount of information exchanged between systems are typically many orders-of-magnitude smaller compared to data exchange between databases. A concrete and well-publicized example is the ARPA-sponsored Knowledge Sharing Effort (KSE) [Patil et al., 1992] which is aimed at developing methodologies and software for the sharing and reuse of knowledge. Among other contributions, the KSE has led to development of an interlingua (KIF) for exchange of knowledge among different computer programs, standards (e.g., KRSL) for unifying the families of frame-based representation languages, as well as high-level protocols (e.g., KQML) for facilitating message-exchanges between different software systems. While these are important advances in their own right, they are somewhat peripheral to the problem we have on hand. For example, the approach taken towards knowledge sharing in Ontolingua (an extension of KIF to facilitate the definition of a common vocabulary for knowledge exchange) is accomplished by a translational approach [Gruber, 1991] which is not unlike a view definition. Since the amount of data that needs to be exchanged is expected to be exponentially larger in our case, this brute-force translational approach can incur unacceptable performance overhead. In [Paquhar et al., 1995], the authors have attempted to appropriate some of the KSE technologies for database integration. Interestingly, they appear to run into the same design choices (e.g., "should there be a shared schema?") with the same tradeoffs. This, in our opinion, is indicative that the KSE is in fact addressing a somewhat orthogonal set of issues.

In summary, we like to point out that the preceding criticisms should be received in perspective. Every one of the above projects presents interesting insights on different aspects of the tremendously complex problem of allowing different data sources to interoperate with one another. Our claim, however, is that virtually all these integration approaches require explicit detection of conflicts by either the user or some other system administrator. (The TSIMMIS approach is one exception, but that is
due to the fact that the system does not detect or resolve conflict automatically.) From this perspective, the Context Interchange strategy is novel and interesting because the system (specifically, the Context Mediator) takes on the responsibility for automatic conflict detection and query mediation based on declarative specifications which captured the meaning of data in different systems.

8.2 Future Work

The advances made in this Thesis have led to a host of new research issues. As interesting and worthy as these may be, current constraints mandate that we postpone them to future work. In the remainder of this section, we present a number of open problems which are either complementary to, or is capable of benefiting from, the formalization described in this Thesis.

First, we observed that 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, additional 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. We noted also that certain groups within the AI community have been actively involved in the construction of domain models or ontologies which presumably provide an ontological basis for knowledge sharing [Lenat and Guha, 1989, Gruber, 1991]. It appears however that ontologies designed for those purposes do not necessarily have all the characteristics which are needed to support the Context Interchange strategy. It would be interesting to examine how the different Knowledge Representation formalisms (e.g., KIF and Ontolingua) can be mapped to our representation and vice versa, and to gain a better appreciation of how these ontologies can be extended so that they can be used in supporting the Context Interchange approach.
Second, we have developed the theory while largely focusing on the problems introduced by semantic conflicts between disparate systems. It is conceivable that one may want to extend the formalism to deal with other problems commonly reported with the integration of information in disparate systems. For example, data present in multiple sources may be *inconsistent* [Agarwal et al., 1995], *incomplete* [Levy, 1996], or *replicated* [Arens et al., 1993]. All of these idiosyncrasies present challenges which warrants further study which may require further extensions to the COIN data model and language which is being proposed in this Thesis. We are optimistic that such extensions will not invalidate the core elements of our data model and framework.

As is pointed in [Lu et al., 1992], the autonomy and heterogeneity of sources present new challenges for query processing and optimization which are not the same as those in distributed database systems. These differences stem from constraints which are characteristic of the underlying environment; for example, different sources may differ in their query-handling ability, cost models may not be known, and data conversions may incur large hidden costs which are not accounted for previously. As we have shown earlier, the detection of unsatisfiable answers in the abductive framework constitute a form of semantic query optimization which presents huge payoffs. We are currently examining how we can take advantage of this framework to deal with more general constraints for identifying queries which can be executed more efficiently. In particular, we have observed commonalities among research findings in abductive reasoning, semantic query optimization and *constraint logic programming* [Jaffar and Maher, 1996] which provide insights on different aspects of the problem. This observation has recently been (independently) reported in [Wetzel et al., 1996].

Yet another fruitful topic for further research consists of identifying tools and techniques for the automatic acquisition of context knowledge corresponding to new data sources for which the context is not known. For example, two data sources may contain data on some overlapping real-world domain: if the context of one of these sources is known, we might apply techniques developed for data mining to discover context rules corresponding to the source with the unknown context. In this instance, the domain model provides a constrained search space for context rules which are
feasible: this greatly enhances the feasibility of identifying nontrivial context rules, which may not have been possible if the search space for these rules is unconstrained.

While constituting a powerful theoretical basis for context mediation, the abduction framework warrants further study. One may pursue this subject at either a pragmatic level or a more theoretical level. As we have pointed out earlier, the SLD+Abduction procedure is a naive procedure and there remains many more opportunities for optimizing its performance. For example, one might exploit known integrity constraints on data sources to interleave abduction with actual databases accesses. In some cases, this can lead to significant improvements in the generation of the mediated query (which incidentally can be much less complex since spurious conflicts may have been eliminated by determining exactly what "kinds" of data are present). At a theoretical level, it would be interesting to provide a characterization of the formal semantics of our abductive framework and to present evidences why it constitutes an appropriate semantics for the problems which we are solving.

Finally, to make the technology we describe usable, there needs to be yet another level of abstraction which insulates users from the complexity of the representational formalism. This involves the development of software tools as well as techniques for creation, consistency checking, modification, merger, and visualization of large repositories of domain and context knowledge. We have embarked on the implementation of several of these tools and are optimistic that the benefits of the mediation technology developed will be more easily transitioned to actual use once these tools are sufficiently robust for actual use. The transition of this technology to a real-world setting will also provide us with the opportunity for the empirical validation of some the claims which we are making. Specifically, it will be interesting to observe how existing representations can be scaled-up or how they can evolve in response to changes.
Bibliography


national Conference on Information and Knowledge Management, pages 629–636, Baltimore, Maryland.


