# Information Integration Using Contextual Knowledge and Ontology Merging

by
Aykut Firat

M.S. Systems Analysis (1998)
Miami University, OH

B.S. Industrial Engineering (1996)
Boğaziçi University

Submitted to the Sloan School of Management
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Management

at the
Massachusetts Institute of Technology
September 2003



Signature of Author: _____
Sloan School of Management
August 27, 2003

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

Certified by:_____
Birger Wernerfelt
Professor of Management Science
Chair, Ph.D. Program, Sloan School of Management

# Information Integration Using Contextual Knowledge and Ontology Merging

by

Aykut Firat

Submitted to the Sloan School of Management on August 27, 2003
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Management

**ABSTRACT**

With the advances in telecommunications, and the introduction of the Internet, information systems achieved physical connectivity, but have yet to establish logical connectivity. Lack of logical connectivity is often inviting disaster as in the case of Mars Orbiter, which was lost because one team used metric units, the other English while exchanging a critical maneuver data. In this Thesis, we focus on the two intertwined sub problems of logical connectivity, namely data extraction and data interpretation in the domain of heterogeneous information systems.

The first challenge, data extraction, is about making it possible to easily exchange data among semi-structured and structured information systems. We describe the design and implementation of a general purpose, regular expression based Caméléon wrapper engine with an integrated capabilities-aware planner/optimizer/executioner.

The second challenge, data interpretation, deals with the existence of heterogeneous contexts, whereby each source of information and potential receiver of that information may operate with a different context, leading to large-scale semantic heterogeneity. We extend the existing formalization of the COIN framework with new logical formalisms and features to handle larger set of heterogeneities between data sources. This extension, named Extended Context Interchange (ECOIN), is motivated by our analysis of financial information systems that indicates that there are three fundamental types of heterogeneities in data sources: contextual, ontological, and temporal.

While COIN framework was able to deal with the contextual heterogeneities, ECOIN framework expands the scope to include ontological heterogeneities as well. In particular, we are able to deal with equational ontological conflicts (EOC), which refer to the heterogeneity in the way data items are calculated from other data items in terms of definitional equations. ECOIN provides a context-based solution to the EOC problem based on a novel approach that integrates abductive reasoning and symbolic equation solving techniques in a unified framework.

Furthermore, we address the merging of independently built ECOIN applications, which involves merging disparate ontologies and contextual knowledge. The relationship between ECOIN and the Semantic Web is also discussed.

Finally, we demonstrate the feasibility and features of our integration approach with a prototype implementation that provides mediated access to heterogeneous information systems.

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

*In memory of Cheng Hian Goh, who passed away on April 1, 1999.*

# TABLE OF CONTENTS

"And the whole earth was of one language, and of one speech…Therefore is the name of it called Babel; because the Lord did there confound the language of all the earth…"

Genesis 11:1-9

# Chapter 1

# Introduction

In the Biblical story of the Tower of Babel, mankind starts to build a city and a tower to stay centralized forgetting the command of God on replenishing the earth. Then, God prevents their endeavor and confuses their speech by introducing multiplicity of languages. Unable to communicate, they terminate the construction and spread all over the earth. Today, thanks to the advances in transportation and information systems, the world has turned into a global village, and we are somewhat able to communicate with each other despite the diversity of our languages and cultures.

A similar story is unfolding in the genesis of computers, in which the confusion of languages already happened, and information systems with multiple languages and assumptions are spread across organizations, and countries. With the advances in telecommunications, and the introduction of the Internet, information systems achieved *physical connectivity*[1], but have yet to establish *logical connectivity*[2]. Lack of logical connectivity is often inviting disaster as in the case of Mars Orbiter, which was lost because one team used metric units, the other English while exchanging a critical maneuver data[3]. Even the ominous events of September 11, could perhaps be prevented, had there been connectivity between the databases of various government agencies including airport security, FBI, and CIA[4].

This problem of attaining logical connectivity among computer systems is traditionally known as achieving *semantic interoperability among autonomous and heterogeneous systems*. In this Thesis, we focus on the two intertwined sub problems of logical connectivity, namely *data extraction* and *data interpretation*, in the domain of heterogeneous information systems.

---

[1] The ability to exchange bits and bytes

[2] The ability to exchange meaningful information

[3] Mars Climate Orbiter Team Finds Likely Cause Of Loss, by Douglas Isbell, Mary Hardin, Joan Underwood, http://mars.jpl.nasa.gov/msp98/news/mco990930.html, September 1999.

[4] Joint Inquiry into Intelligence Community Activities before and after the Terrorist Attacks of September 11, 2001, by The House Permanent Select Committee On Intelligence And The Senate Select Committee On Intelligence, December 2002.

The first challenge, data extraction, is about making it possible to easily exchange data among semi-structured and structured information systems. Web sites, for example, contain huge amounts of data, yet operational systems cannot easily use them because of the heterogeneity in the protocols used to reach and extract data (e.g. SQL vs. http). As described in this Thesis wrappers can be used to overcome this problem, by providing an artificial (and usually third-party) interface to the data sources.

The second challenge, data interpretation, deals with the existence of heterogeneous contexts, whereby each *source* of information and potential *receiver* of that information may operate with a different context, leading to large-scale semantic heterogeneity. A *context* is the collection of implicit assumptions about the context definition (i.e., meaning) and context characteristics (i.e., quality) of the information. As a simple example, whereas most US universities grade on a 4.0 scale, MIT uses a 5.0 scale. Another typical example might be the extraction of price information from the Web: but is the price in Dollars or Yen (if dollars, is it US dollars or Hong Kong dollars), does it include taxes, does it include shipping, how current is it, - and most importantly does that match the receiver's assumptions? With the global reach of the Internet, contexts of data sources are no longer obvious to their users: they have to be declared and exchanged together with the data, and be reconciled whenever a conflict exists. The Context Interchange (COIN) group at MIT has investigated the existence of and reasons behind various forms of context challenges and developed a strategy, and theory for representing context knowledge, and a *context mediation* engine for mitigating the problem.

COIN *strategy* was inspired from earlier work reported in [Siegel and Madnick, 1991, Sciore et al., 1994], and later studied by Cheng Hian Goh in his Ph.D. thesis [Goh 1997], which introduced the formal definition of a COIN *framework*. COIN strategy rests on the notions of *context* that allows users to furnish a logical specification of how data are interpreted in sources and receivers, and *conversion functions*, that specifies how conflicts, when detected, should be resolved. This approach is fundamentally different from classical integration strategies, as it does not insist on users or system administrators to determine what conflicts exist a priori between any two systems.

While Goh's study was an important first step towards solving the problem of interoperability among heterogeneous systems, it also left out a number of important topics and problems. First, and foremost, there was not a clear definition of concepts such as *context, conversion function, and ontology*. Second, the COIN framework was unable to deal with many types of heterogeneities that surfaced after working with several industry information-providers in attempting to apply the COIN *technology* to the "real world" problems encountered by them. COIN framework, for example, was silent on *equational ontological conflicts* (EOC) that refer to the heterogeneity in the way data items are calculated from other data items in terms of definitional equations. Third, Goh's study did not address merging independently developed, ontology based COIN *applications*. Finally, there have been significant developments in the recently emerging Semantic Web research, many of which have important implications for database integration. There is a need to explain the relationship between the COIN strategy and Semantic Web efforts and to exploit the synergies between them.

## 1.1 Summary of Contributions

The primary objective of this Thesis is to provide approaches and theory --coupled with a robust and flexible software platform-- to the data extraction and interpretation problems, thereby contributing to the solution of the *semantic interoperability among autonomous and heterogeneous systems* problem.

In the data extraction part of this Thesis, we introduce a technology and infrastructure to support the effective flow of information among sources and services on the web and their interconnection with legacy systems that were designed to operate with traditional relational databases. This technology, named Caméléon [Firat et. al 2000], is designed to work as a relational front-end to semi-structured data sources as well as traditional relational databases. It can extract data from web pages using declarative specification files that define extraction rules. We use regular expressions in defining extraction rules, that segment and iteratively extract attribute data values. The users can then issue SQL queries and treat web sites as if they were traditional databases. This allows software that can open connections to Web, (e.g. Excel, Visual Basic, etc.), as well as traditional user application software to directly query the Web. As a separate application, using a post-processor, this technology has also been used to generate XML-tagged pages from "legacy" HTML Web sites.

The rest of the contributions are in the data interpretation part of this Thesis. First, we extend the existing formalization of the COIN framework with new formalisms and features to handle larger set of heterogeneities between data sources. This extension, which will be referred to as Extended Context Interchange (ECOIN) framework from now on, is motivated by our analysis of financial information systems that indicates that there are three fundamental types of heterogeneities in data sources: *contextual, ontological, and temporal*.

ECOIN framework transforms ontological heterogeneities (i.e. differences in the definitions) into contextual heterogeneities, thus builds on top of the existing COIN model. In particular, we are able to reason with *equational* ontological conflicts by extending the reasoning engine with symbolic equation solving capabilities. Consider for example, financial concepts such as “profits after taxes” and “profits before taxes” that are ontologically distinct but have interdependences that can be expressed as equations, such as “profits after taxes = profits before taxes – taxes”. Such conflicts in accounting methods are quite widespread not only between different countries, but also within the same country [Firat et al. 02]. For example, The Wall Street Journal and S&P use different methods to calculate the P/E Ratios for the Standard & Poor's 500-stock index. The Wall Street Journal divides the combined market capitalization of the 500 companies currently in the index by their most recently reported four quarters of earnings, while S&P updates earnings statistics for the index just once a quarter and doesn't revise earnings from previously reported quarters to account for additions or deletions to the index.[5] Therefore, this extension by itself covers a wide range of problems.

[5] Moving Target: What's the P/E Ratio? Well, Depends on What Is Meant by Earnings --- Terms Like “Operating”, “Core”, “Pro Forma” Catch Fire, Leave Investors Muddled --- “Earnings Before Bad Stuff', Jonathan Weil, Wall Street Journal, Aug 21, 2001.

ECOIN provides a context-based solution to the EOC problem by making the context of the data items of each source explicit (i.e., how they are derived from other data items) and adjusting their values to different contexts by recalculating them when necessary using the contextual knowledge – including the definitional equations associated with each context. Equational ontological conflicts are not handled by making changes to the ontology, for example by introducing new types and defining equational relationships between their values. Making changes in ontologies is likely to be a time-consuming and difficult process, and is better avoided as much as possible. Furthermore, in many cases, such an approach would result in an explosion of new ontology types to handle all of the possible variations. In ECOIN framework we use modifiers, a special type of attributes that collectively define the context of a data source, to specify the implicit aspects of an ontological term. We claim that ECOIN approach is an elegant and low cost way to represent equational ontological conflicts.

Second, query mediation in ECOIN, is a novel approach that integrates abductive reasoning and symbolic equation solving techniques in a unified framework. We build on top of the abductive inference approach of COIN framework with the addition of symbolic equation solving capabilities by using constraint logic programming techniques. Our combination of symbolic equation solving with abductive reasoning constitutes an interesting example for the emerging work under the name *abductive constraint logic programming (ACLP)* [Kakas et. al 2000]. We think of symbolic equation solving axioms as constraints to be satisfied by the abducted equational answers. Similar to how integrity constraints behave in COIN query processing, equational constraints simplify, combine and transform the abducted equational answers. In addition consistency checks for equations can sometimes prune query branches and demonstrate what is traditionally known as *semantic query optimization*.

Our choice of solving equations symbolically as opposed to transforming the data at run time is consistent with query processing in COIN framework, which constructs the *intensional answers* as opposed to *extensional answers* that would be obtained during run time. ECOIN, like COIN, accepts a *naïve query (*i.e. query with the assumption that no conflicts exist between source and receiver contexts) and rewrites it into a *mediated query* (i.e. query with all conflicts between sources and receivers reconciled) with the extra processing capability of equations that originate from the conversion functions used to transform query terms from one context to another.

Third, we address the merging of independently built ECOIN applications, so that queries covering multiple applications can be answered. From the user point of view, it is usually more advantageous to merge already existing applications with their accompanying ontologies, instead of creating a new application with a broader ontology from scratch. The challenge of merging multiple applications lies in the existence of modeling differences between independently developed ontologies and the emergence of new contextual conflicts because of using different applications together. We adopt a virtual context centered approach to merging ECOIN applications. We call it virtual because we do not create a materialized application from the applications to be merged. Instead, we create an application with articulation axioms defining the relationships between the context definitions, and related ontology elements. We call it context centered, because the motivation behind the merging is to achieve the exchange of contexts between different applications. Since our goal is to be able to answer queries

that cover multiple applications, it is sufficient to relate context definitions between applications rather than linking all ontology elements. In order to avoid the cluttering of articulation axioms, we adopt a hierarchical approach based on merging two applications at a time. The merger application becomes an independent application by itself, and the user need not be aware of the fact that it is a virtual application created by merging other applications. This approach with its simplicity and requirement of minimal articulation axioms constitutes a powerful approach to merging independently developed ECOIN applications for query answering purposes.

Fourth, we initiate the process of finding a mapping between the ECOIN framework and the Semantic Web. As pointed in [Manola 2002], much of the Semantic Web research activity is taking place in areas somewhat separated from the traditional database and information systems communities. One of the main reasons of this separation is the insufficient articulation of the relevance of heterogeneous database research to Semantic Web research. One of our aims in this Thesis is to explain how COIN strategy is relevant to Semantic Web. To make this relevance concrete, we discuss mappings from our internal representation of ECOIN framework to Semantic Web languages such as RDF, RDFS, and OWL.

Finally, we remark that ECOIN framework has been realized in an actual prototype implementation demonstrating the feasibility and features of our approach. This prototype provides mediated access to traditional databases, as well as semi-structured web sites, and web services; creates and maintains metadata (e.g. ontologies, context descriptions) that are used in ECOIN through graphical interfaces, and supports merging multiple applications.

## 1.2 Thesis Outline

The rest of this Thesis is organized as follows. **Chapter 2** is a self-contained study of data extraction in which we describe the Caméléon approach to the interoperability of Web sources and traditional relational databases. This chapter aims to provide a flavor of infrastructural issues that have to be dealt with before going into higher level issues related to data interpretation in the following chapters. The chapter starts with an analysis of different approaches to data extraction and compares our approach with the existing approaches in the literature. Then, the architecture of Caméléon and the structure of declarative *spec files* that describes schema and extraction rules are explained in detail. We provide an airfare aggregation example to illustrate the advanced features of Caméléon web wrapper engine. Finally, we discuss the architecture and features of next generation data extraction tools based on the latest developments (e.g. XML and Web Services).

**Chapter 3** delves into the data interpretation aspects of information integration by first categorizing the dimensions of data heterogeneity as *contextual, ontological, and temporal* based on a case study we conducted in a financial setting. Then we provide a literature survey on data heterogeneities and major approaches to achieving semantic interoperability among autonomous and heterogeneous systems. The objective of this chapter is to familiarize the reader with existing approaches to information integration, which will hopefully lead to a better understanding of the subtleties of our approach described in the following chapters.

Our aim in **Chapter 4** is to explain what has already been done within the COIN group, and what new contributions we are offering with this Thesis. We provide a comprehensive summary of the COIN approach for readers who may choose to skip the eloquent but detailed presentation in [Goh 97]. We use illustrative examples to explain the features of the existing COIN approach, and to underscore the differences between ECOIN and COIN. We end the chapter with a comparison of ECOIN and COIN to highlight our contributions that extends and complements the previous approach in our group.

**Chapter 5** is devoted to explaining the core concepts of ECOIN approach, and laying out the complete ECOIN data model. We aim to convey the philosophy behind the central constructs of ECOIN such as *context* and *ontology* by summarizing insightful works from the literature. We then position these concepts in the formal ECOIN data model, which culminates in the description of ECOIN framework. ECOIN framework specifies a template that can be used to integrate autonomous and heterogeneous data sources. Our formal description of the ECOIN data model is aided by examples, and intuitive explanations. The reader interested in implementing the ECOIN approach will hopefully find formal statements precise enough, whereas the reader who is interested in a high level understanding may generate the big picture from intuitive explanations and examples.

**Chapter 6** focuses on query answering in the presence of *equational ontological conflicts* using *abductive constraint logic programming (ACLP)*, which is the primary difference between ECOIN and COIN. We first clarify what we mean by equational ontological conflicts and then present the theoretical foundations of *abductive* and *constraint logic programming* paradigms. Our aim in this chapter is to describe how ACLP provides an elegant way for query mediation in the ECOIN framework. In particular, we focus on the representation of a simultaneous symbolic equation solver in constraint logic programming which integrates nicely with the abductive logic programming framework already employed in the COIN framework. We proceed to explain the building blocks of symbolic equation solving using *constraint handling rules* and its interaction with abductive inference during query processing. The chapter ends with an example that illustrates how a sample query involving equational ontological conflicts is mediated. The material in this chapter is crucial for readers who are interested in building an inference engine that can mediate SQL queries with the help of metadata from the ECOIN framework.

In **Chapter 7**, we consider how one could proceed to merge multiple ECOIN applications, which involves merging disparate ontologies and contexts. We begin with a review of ontology merging, and schema integration literature to provide the relevant background in this area. Next, we extend our airfare example from Chapter 4 with a car rental application and illustrate the concept of merging using these two applications. After algorithmically explaining how virtual context centered merging works, we formally describe the incremental elements of our merging framework. Our primary aim in this chapter is to demonstrate the extendibility of our integration approach with context driven merging, which is low cost and scalable.

In **Chapter 8** we present the ECOIN prototype that demonstrates the feasibility of ideas described in previous chapters. While the chapter describes all three processes--the client, mediation and server processes-- that make up the prototype, it focuses on the

implementation of mediation, particularly the abduction algorithm. Abduction is explained with a step by step evaluation of a query sub-section. The treatment of client and server processes is brief and refers readers to detailed works of several master students.

We discuss the relationship between ECOIN and Semantic Web in **Chapter 9**, and explore mappings between the two frameworks. In particular, we consider the relationships between ontologies, and the representation of context on the Semantic Web. We end this chapter by mentioning future directions for research that relates ECOIN and the Semantic Web.

Finally, we conclude in **Chapter 10** by pointing out some promising research areas to pursue in the future.

# Chapter 2

# Data Extraction

With the advent of the Internet, the volume of on-line data has skyrocketed. Yet, much of these data are structured primarily for human consumption, and it is difficult for computer programs to gather, and operate on data that do not have a *well-defined* and *agreed-upon* structure designed for machines. Luckily, it is possible to discover some level of structure by analyzing the data prepared for human consumption. Data extraction is about artificially imposing a well-defined structure over semi-structured data [Abiteboul 97]), therefore enabling the exchange of data among heterogeneous types of information systems (e.g. SQL vs. http).

In this part of this Thesis, we introduce a technology and infrastructure to support the effective flow of information among the sources and services on the Web and their interconnection with legacy systems that were designed to operate with traditional relational databases. This technology, named Caméléon, is designed to work as a relational front-end to semi-structured data sources. It extracts data from web pages using declarative specification files (*spec file* for short) that define extraction rules expressed in regular expressions. The users can then issue SQL queries to Caméléon and treat web sites as if they are traditional databases. This allows ODBC-compliant package software, such as Excel, Visual Basic, etc., as well as traditional user application software to directly query the Web. As a separate application, using a post-processor, this technology has also been used to generate XML-tagged pages from "legacy" HTML Web sites.

We start with a review of literature related to data extraction. Our aim is to provide the necessary background to ensure the smooth flow of ideas in this Thesis, while aiding the reader to understand where and how this work differs from similar studies. Then, we go into the details of Caméléon wrapper engine design and implementation.

## 2.1 Literature Review

During the first boom years of the Internet, especially with the emergence of aggregators [Madnick and Siegel 02], there has been a proliferation of data extraction technologies, often-called *Web wrappers (or wrappers for short)*[6], to absorb Web data

[6] Also called “screen scrapers” based on similarity to technologies of the 1980’s and 1990’s

into the information food chain. These wrappers, developed both by industry and academia (see [Firat et. al 00] for a list), either support rich query languages such as SQL or OQL ([Roth & Schwarz 97], [Abiteboul 97]) to query the web sources, or emphasize conversions from HTML to XML (e.g. XWRAP [Liu et. al 99], W4F [Sahuguent & Azavant 99]), thus making the aggregation of Web sources easier.

A typical web wrapper responds to some type of query by retrieving a web page, applying extraction rules specified in a *specification file* corresponding to the query, and presenting the extracted data in a structured format. The problem of creating a wrapper is defined more precisely in [Laender et. al 2002], as follows:
*"Given a Web page S containing a set of implicit objects, determine a mapping W that postulates a data repository R with the objects in S. The mapping W must also be capable of recognizing and extracting data from any other page S' similar to S."*
Wrappers retrieve web pages by using a web client, whose capabilities have great practical importance. A comprehensive web client supports:

- http methods such as get and post,
- responses to standard HTML headers such as automatic refreshes and redirections,
- automatic cookie handling,
- secure socket layer (SSL),
- authentication,
- certificates, and
- interpreting script languages such as JavaScript.

Ideally, it would be best to employ http clients used in browsers such as Internet Explorer and Netscape. They are not, however, always exposed to the public completely, or their use is not convenient in every programming environment. Mozilla, .NET and Java libraries are some popular choices, which may be turned into comprehensive clients with some extra programming.

Wrappers treat Web pages either as a document tree or as a data stream. Wrapper engines like W4F [Sahuguent & Azavant 99], and Lixto [Baumgartner et. al 01] parse Web pages using Document Object Model (DOM)[7] into a tree, and the extraction rules are expressed primarily in terms of the DOM. Other wrapper engines such as TSIMMIS [Garcia-Molina et. al 95] and Caméléon [Firat et. al 2000] ignore the HTML tag-based hierarchy and treat Web pages as a sequence of characters. Extraction rules in this category are usually expressed in terms of regular expressions. Advantages and disadvantages of these two approaches are shown in Table 2.1 and Table 2.2 below.

In the *Web pages as a document tree (WAD)* approach, HTML pages are parsed into a document tree, and the hierarchical relations between different HTML elements are preserved. In WAD approach, it is easy to construct extraction rules by using the DOM. For example, one can refer to the contents of first row of the first table in an html document with "*doc.table[0].tr[0].text*". The major disadvantage of this approach is that more than 80% of the HTML pages do not conform to the HTML standard, and extra tools such as HTML Tidy[8] are needed [Sahuguent & Azavant 99] to mitigate the problem. In addition, when HTML and DOM are extended with new elements, parsers have to be updated, thus increasing the maintenance cost of systems built this way.

[7] http://www.w3.org/DOM/
[8] http://www.w3.org/People/Raggett/tidy/

Finally, we should mention that parsing HTML pages is an expensive operation, which may affect the performance adversely.

| ADVANTAGES | DISADVANTAGES |
|---|---|
| **Easy rule construction:** Extraction rules are easy to construct by utilizing the HTML tag hierarchy. **Concise rules:** Allows powerful yet concise extraction rules by the using the DOM model. **Preservation of Hierarchy:** Associations among hierarchical elements (i.e. column name and column elements) are possible. | **Irregularity:** Less than 20% of the pages are conforming to the HTML standards; therefore HTML parsers need error recovery mechanisms. **Currency:** Parser has to be updated with changing HTML versions. **Performance:** Parsing HTML into a tree is expensive. |

Table 2.1. Advantages and Disadvantages of Web Pages As A Document Tree Approach

| ADVANTAGES | DISADVANTAGES |
|---|---|
| **Generality:** Independent of HTML, thus not affected by the Web languages. It would work just fine with XML or anything else. **Granularity:** Matching with any level of granularity is possible. **Performance:** Regular Expression pattern matching is significantly faster than parsing. | **Complexity:** Regular Expressions are harder to form and understand by regular users compared to HTML as a document tree approach. **No-nesting:** In order to be efficient regular-expression pattern matching does not backtrack. **Limited Association:** Hard to associate hierarchical relations, i.e. inferring column elements from the column name or number. |

Table 2.2 Advantages and Disadvantages of Web Pages As A Data Stream Approach

In the *Web pages as a document tree (WAD)* approach, HTML pages are parsed into a document tree, and the hierarchical relations between different HTML elements are preserved. In WAD approach, it is easy to construct extraction rules by using the DOM. For example, one can refer to the contents of first row of the first table in an html document with "*doc.table[0].tr[0].text*". The major disadvantage of this approach is that more than 80% of the HTML pages do not conform to the HTML standard, and extra tools such as HTML Tidy are needed [Sahuguent & Azavant 99] to mitigate the problem. In addition, when HTML and DOM are extended with new elements, parsers have to be updated, thus increasing the maintenance cost of systems built this way. Finally, we should mention that parsing HTML pages is an expensive operation, which may affect the performance adversely.

The *Web pages as a data stream (WAS)* approach treat Web pages as a sequence of characters. Wrappers can be generated not only for HTML pages, but also for other text data sources including XML pages, and e-mail documents. Mostly, regular expressions are used in specifying extraction patterns, which increases the power of pattern specification. Unlike the pure DOM model approach whose granularity is limited by the granularity of HTML elements, regular expressions can be used to specify patterns at any

level of granularity. For example, while one cannot express the “first two digits in bold in the first row of the first table” using DOM, it is possible to express the same thing with regular expressions as “.*?<table>.*?<tr>.*?(\d\d).*?”. Regular expressions based pattern matching is fast, but expressions are difficult to form and understand for beginners.

Wrappers can also be classified into three categories based on how their specification files are generated: manual, semi-automatic and automatic. A brief survey on this classification with additional pointers can be found in [Tatbul et. al 2000]. In the manual approach (e.g. Jedi [Huck et. al 98]), users create general extraction rules by analyzing a representative set of web pages, and they are responsible for updating the specification files when necessary. In automatic generation, users first have to annotate a number of training examples through a visual interface (e.g. SoftMealy [Hsu and Dung 98]). Machine learning algorithms, such as inductive learning, are then applied to generate the specification files (e.g. Wien [Kushmerick et. al 97], Stalker [Muslea et. al 99]). Semi-automatic approaches do not use any machine-learning algorithms but try to make the spec file creation easier through mappings between the visual and text or DOM views, by making suggestions on patterns that need to be approved or modified by the user. Manual approaches are known to be tedious, time-consuming and require some level of expertise concerning the wrapper language. In addition, when web sites change, updating of specification files have to be done manually as well. Given the state of the art in automatic wrapper creation, however, manual and semi-automatic approaches are currently better suited for creating robust wrappers than the automatic approach. The maintenance costs of current automatic approaches are also comparable to manual and semi-automatic approaches, since in the automatic approach the user has to annotate new training samples when the wrapped web pages are modified. In fact, as noted by [Knoblock et al. 00], it is unrealistic to assume that a user is willing and has the skills to browse a large number of documents in order to identify a set of informative training examples. While new approaches are being suggested that require a small number of training samples [Knoblock et al. 00], their applicability is limited to simpler Web pages that do not contain various sorts of exceptions. On difficult web pages the lack of informative examples would lead to low accuracy.

A third grouping can be made on whether the wrappers are declarative or not. In this context, declarative means that there is a clean separation of extraction rules from the computational behavior of the wrapping engine. Non-declarative wrapper engines mix extraction rules with a programming language (e.g. W4F with Java) or offer a programming language of their own (e.g. Compaq’s WebL). Figure 2.1 shows an example of a non-declarative W4F specification file created for CIA fact book. In declarative wrapper engines, extraction rules are separated from the computation logic and do not require any compilation of the rules into executable code. In Figure 2.2, we show such an example, a logical description of the data to be extracted from an eBay page for the Lixto wrapper engine.

In Table 2.3, we map the existing academic wrapper engines into the three meta-categories we discussed above: whether the wrappers are declarative or not; whether they view Web pages as a document (WAD) or as a data stream (WAS), and whether the wrapper creation is manual, semi-automatic or automatic.

```
SCHEMA{ String capital;}
EXTRACTION_RULES{
   capital = html.body.p[i].b[0]->pcdata[1].txt
   where html.body.p[i].b[0].txt =~ "National capital";
}
RETRIEVAL_RULES{
 getCountry(String ciaCode){
   METHOD: GET ;
   URL: "http://www.odci.gov/cia/publications/factbook/$ciaCode$.html";
   }
}
JAVA_CODE{
 public static void main(String args[])
   throws Exception{
     CIA_Country country = CIA_Country.getCountry("fr");
     System.out.println(country);
   }
}
```

Fig. 2.1. W4F Extraction Rules for CIA Fact book (attribute Capital)

| | |
|---|---|
| ebaydocument(S, X) | ←getDocument(S = $1, X). |
| tableseq(S, X) | ←ebaydocument(_ , S),<br>subsq(S, (*.body.*.center, []), (.table, []), (.table, []), X),<br>before(S, X, (*.tr, [(elementtext, Current, substr)]), 0, 0,_ , _ ),<br>after(S, X, (*.img, [(src, spacer.gif, substr)]), 0, 0,_ , _ ). |
| record(S, X) | ←tableseq(_ ,S), subelem(S, .table, X) |
| itemdes(S, X) | ← record(_ , S), subelem(S, (*.td. * .content, [(href, , substr)], X) |
| price(S, X) | ← record(_ , S), subelem(S, (*.td, [(elementtext, \var[Y]._, regvar)]), X),<br>isCurrency(Y). |
| bids(S, X) | ← record(_ , S), subelem(S, *.td, X), before(S, X, .td, 0, 30, Y, ),<br>price(_ , Y) |
| date(S, X) | ← record( , S), subelem(S, *.td, X), notafter(S, X, .td, 100) |
| currency(S, X) | ← price(_ , S), subtext(S, nvar[Y], X), isCurrency(Y) |
| pricewc(S, X) | ←price(_ , S), subtext(S, [0 _ 9]+\.[0 _ 9]+, X). |

Fig. 2.2 Elog Extraction Rules for a a single eBay page

(adopted from [Baumgartner at. al 01])

| | **Declarative** | | **Non-Declarative** | |
|---|---|---|---|---|
| | ***WAD*** | ***WAS*** | ***WAD*** | ***WAS*** |
| **Manual** | | Mobie(Tsimmis) | Jedi | Araneus, WebL |
| **Semi-automatic** | NoDoSe | Caméléon | W4F | |
| **Automatic** | Lixto | WIEN, Stalker | XWrap | |

Table 2.3. Classification of Web wrapper projects[9]

Commercial wrapper engines are not as easily analyzable as the academic ones as they usually require purchase of the system. For that reason, we will only provide a list of these wrapper engines in Table 2.4.

[9] This list primarily covers systems whose source codes were available for testing

| Company | Tool |
|---|---|
| AT&T | Whirl |
| Connotate Technologies | vTag |
| Crystal Software | TextPipe |
| Data Junction | Content Extractor |
| Extradata Technologies | Unwwwrap |
| Fetch Technologies | AgentBuilder |
| firstRain | firstRain Studio |
| IBM | Garlic, Intelligent Miner |
| ItemField | ParserStudio |
| Kapow Technologies | RoboSuite |
| Knowmadic | WebActivity Integration Suite |
| Lencom Software | Visual Web Task |
| Lixto | Lixto Visual Wrapper,<br>Lixto Transformation Server |
| Loton Tech | WebDataKit |
| Orsus Solution | UnoStudio |
| QL2 Software<br>(Formerly: Caesius Software) | WebQL<br>(Web Query Language) |
| Republica | X-Fetch Wrapper |
| Sagent | ETL |
| ShueTech | Mine The Web |
| Temis Group | Online Miner, Insight Discoverer Extractor |
| Thunderstone | Webinator |
| WebMethods | Integration Platform |
| XSB | Xrover |
| Yodlee | Yodlee |

Table 2.4 Commercial Wrapper Products[10]

## 2.2 Caméléon Wrapper Engine

Data extraction research in the COIN group dates back to 1996 and earlier. The first wrapper engine, Generic Screen Scraper (GSS) was developed in Perl using regular expressions and finite state automata [Qu 96]. Later Jakóbisiak implemented a wrapper generator with minor changes and constructed a multidatabase browser to provide a single query interface to heterogeneous sources [Jakóbisiak 96]. Then came Grenouille [Bressan & Bonnet 97] with a slightly different approach. In Grenouille regular expressions applied to the whole page and were defined for a tuple. Later Grenouille was converted from Perl to Java keeping the same design [Ambrose 98].

[10] Part of the table is adopted from http://www.wifo.uni-mannheim.de/~kuhlins/wrappertools/

In 2000 we developed Caméléon in Java based on a new design, which will be explained in detail in this section. Finally, in 2003, we moved from Java to C# taking advantage of the tools provided in .NET library and made some minor changes to the Caméléon engine. From a design and implementation perspective, Caméléon, is superior to all previous efforts in our group, and was in fact licensed out by MIT to a technology start up.

### 2.2.1 Caméléon Architecture

Caméléon is a wrapper engine with a dual personality. To web servers it is like any other Internet browser; to its users it is like a relational database system with some restrictions. There are two major components that make Caméléon a *virtual relational database*: the relational query front-end, and the core data extraction engine. Relational front-end consists of a planner, optimizer, and an executioner (POE), which brings major performance improvements with the parallel execution of multiple Web queries. We will leave the details of POE to [Alatovic 02], and suffice it to say here that it takes an SQL query, creates a plan considering the capability declarations of the sources, optimizes the plan based on cost characteristics, and then executes sub-queries using the core Caméléon. In this section, we focus on the design and implementation of the core engine.

The core of Caméléon is composed of the *query handler*, *extraction*, and *retrieval* modules as shown in Figure 2.3. Based on input queries in SQL and interaction with a registry[11] the query handler determines which spec file needs to be retrieved and where it should look for those files, which attributes need to be extracted, and how to display the output. The Query handler module uses a spec-parser to validate and parse a spec file. In the Java version of Caméléon, we implemented the spec-parser using a compiler-compiler language (i.e. javacc and jjtree), and in C# implementation we shifted to XML-based spec-files, thus utilized the built-in XML parser and X-Path expressions.

The scope of the extraction module is limited to applying extraction rules to a data stream. As we mentioned before, some wrapper engines treat Web data as a document tree and utilize special-purpose parsers coupled with cleaning tools like Tidy at this stage. Since Caméléon treats web data as a sequence of characters, its extraction module is simpler and only responsible for executing regular expression patterns against web data. In the Java version, we utilized a third party regular expression engine (OroMatcher) to implement this module, whereas in the C# version we adopted the .NET library for regular expressions.

The retrieval module is perhaps the most important module of a wrapper engine, as its capabilities constrain the range of Web pages a wrapper can fetch. In Caméléon, the retrieval module deals with get and post methods, authentication, redirection, cookies, and SSL, therefore maximizing the range of accessible Web pages. In the Java version we utilized a third party web client (HTTPClient) with some modifications, whereas in the C# version we employ the built-in .NET web client with some extensions. Retrieval module is also responsible for interpreting script languages such as Java Script, which may be essential in retrieving a Web page (e.g. when a cookie is set through a Java Script). The C# version capitalizes on the language independence feature of .NET and directly invokes the Microsoft script interpreters to emulate the essential script operations

[11] Registry is a collection of metadata, analogous to the catalog table of traditional database systems.

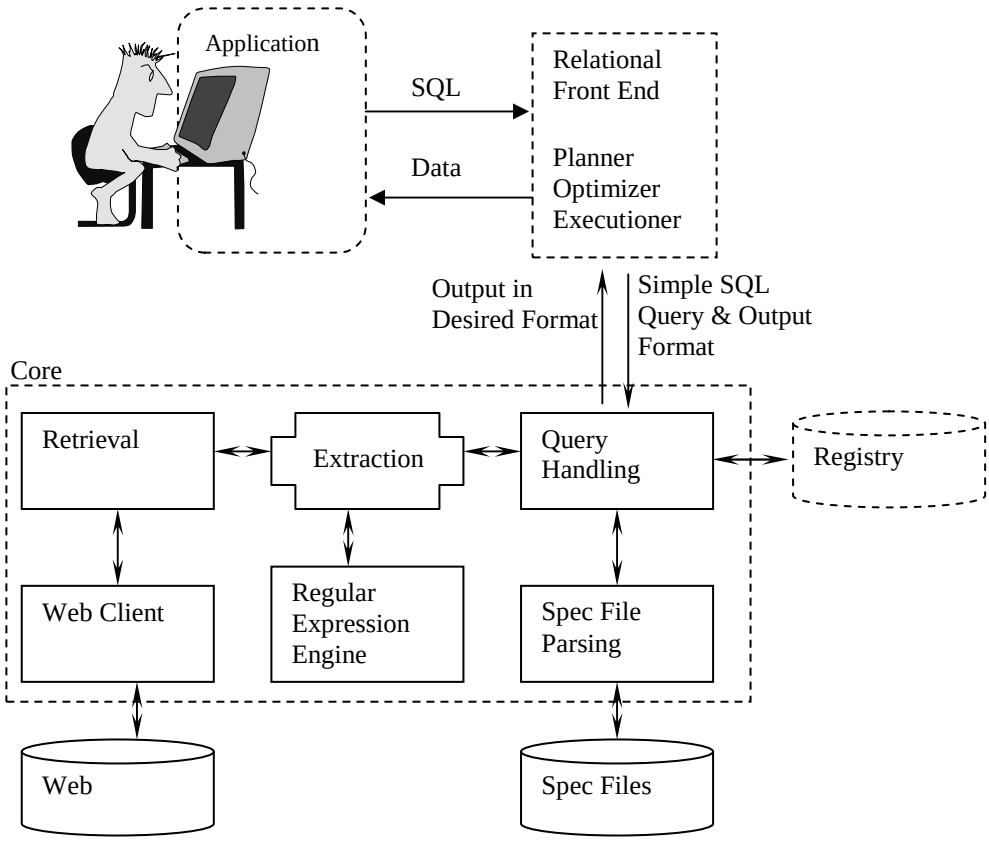



Figure 2.3. Caméléon Architecture

in web pages. Furthermore, the retrieval module is able to follow a number of pre-requisite web pages, (e.g. to obtain a link, to store cookies, supply authentication information, etc.), before arriving to the desired page.

## 2.2.2 Caméléon Spec File Structure

One of the important properties of a desirable wrapper engine is the simplicity and expressiveness of its specification language. Logic based specification languages such as Elog are highly expressive but they are not easy to understand. In Caméléon spec files, we aimed to balance expressiveness with simplicity. Having wrapped hundreds of web sites in the past years, we believe that Caméléon spec file language is easy to learn, and its expressiveness is satisfactory for the vast majority of cases.

Each Caméléon spec file can be thought of as a relation or table in the virtual database of Web. Patterns in a Caméléon spec file are based on the simple idea of first segmenting a Web page, then applying a pattern to extract the data values within that region. Consider, for example, the following example pattern specification to extract "Coordinates" attribute from the cia fact book web page:

```
<ATTRIBUTE name="Coordinates" type="String">
<BEGIN>Geographic\s*coordinates:</BEGIN>
<PATTERN><td[^>]*><font[^>]*>\s*([\0-\377]*?)\s*<</PATTERN>
<END></tr></END>
```

</ATTRIBUTE>[12]

The specification above is given in terms of XML, which is used in the .NET version of Caméléon, and starts with the *attribute* tag specifying the name (Coordinates) and type (String) of the attribute to extract. The following *begin* and *end* tags identify a sub region in the page. This region is the segment between the indexes of the pattern matches after applying the *begin* and *end* patterns to the page sequentially--*end* pattern applying after the index of begin pattern match. The pattern specified in the *pattern* declaration applies to this region as many times as possible, and the part of the pattern match designated by the enclosing parentheses is saved as an attribute value. The notion of regions makes it easier to create patterns by limiting their scope. Complex regular expressions are applied only to a small portion of the page, which increases the efficiency of the extraction.

One problem with the above approach of extracting attribute values independently is identifying the tuples after all data is extracted (i.e. how should the attribute values be merged to obtain the records). We make two assumptions in the pattern specification that solves this tuple identification problem. First, we assume that the pattern matches will be ordered, i.e. if two attributes have equal number of extracted data values then the $i^{th}$ value of attributes will be merged to obtain the $i^{th}$ record. This makes tuple identification trivial when the match numbers across attributes are equal. Second, we assume that the $j^{th}$ (index starting at 0) data value of an attribute with k data values is calculated as *j modulo k*. This provides a match-up when the extracted data value numbers for attributes are different.

In Figure 2.4, we present a complete spec file for yahoo travel Web site. Some of the features shown in Figure 2.4 are explained in detail in the following subsection that summarizes the features of Caméléon spec files. As seen in Figure 2.4, we express spec files using XML to benefit from the availability of XML parsing tools. A spec file starts with a relation name declaration that will be used to refer to the data elements defined by that spec file. Then one or more source declarations with their attribute extraction rules are defined. In Figure 2.4 we show some of the advanced features of a spec file such as using the post method, parameter replacement, prefix and suffixes. These and other features of Caméléon are explained next.

## Spec Files

### 1. Disjunction

Sometimes it is not possible to discover a single pattern that would match the desired data across all similar pages. In these types of cases we allow disjunctive patterns to specify multiple patterns. The following is an example of such a situation in which two disjunctive patterns are specified for a single attribute.

```
<ATTRIBUTE name="LastTrade " type="String">
<BEGIN><![CDATA[Last\s*Trade]]></BEGIN>
<END><![CDATA[</TR>]]></END>
<PATTERN><![CDATA[<B>\s*(.*?)\s*<FONT\s*SIZE=1>(.*?)</FONT>]]></PATTERN>
<PATTERN><![CDATA[<B>\s*(\d+)\s*</B>]]></PATTERN>
</ATTRIBUTE>
```

[12] The pattern specification is simplified by excluding Cdata elements.

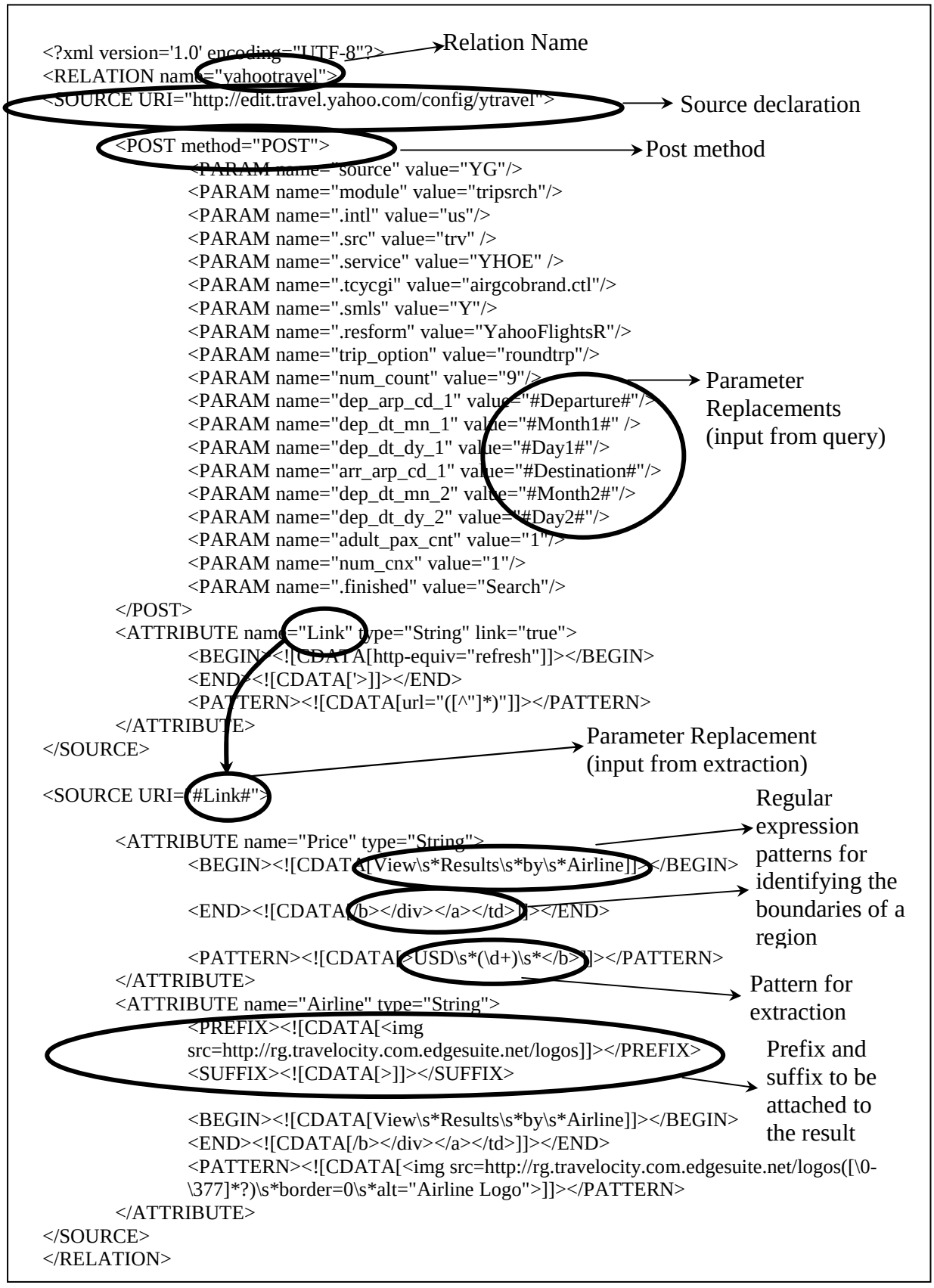



Figure 2.4 Caméléon Spec-File for Yahoo Travel

We should note, however, that these disjunctive patterns are not mutually exclusive and occasionally special care must be taken to construct patterns whose intersections are empty. Otherwise the same item will be matched multiple times and repeated in the output.

**2. Conjunction**

In spec files it is possible to define conjunctive patterns, by simply denoting them with enclosing parentheses. The semantics of conjunctive patterns in Caméléon corresponds to the concatenation of pattern matches. The first pattern element in the example above that extracts stock prices has such a case with two groups of enclosing parentheses, the first one matching the whole part of the last trade value, the second one the fractional part. The matched elements are then concatenated to form a single price value. This feature is very useful, when data to be extracted is not atomic, and separated by unwanted tags.

**3. Multi-page transitions**

When wrapping web pages, we sometimes need to traverse multiple pages to locate the page we want to extract information from. This situation occurs when the URLs are created dynamically (e.g. a session ID is assigned for each access to the page), or a cookie needs to be established before you can go to the desired page, or there is no simple way of deducing the desired URL without visiting a particular page, or the data is spread through multiple pages. To handle these kinds of cases Caméléon has a feature that lets us wrap multiple pages for a relation.

In Figure 2.4 for example the link to the second page is extracted from the first web page. We need to perform this step in this case because there is no simple way of deducing in advance what the link is supposed to be. Then the link is supplied to the next source element, which takes us to the page where we want to extract the values of price and airline.

**4. Parameter Replacement**

Parameter replacement is the use of input or extracted attribute values within the subsequent elements in the spec file. In the multiple page traversal case, we have seen one example of this. The value of attribute “Link” was used in the next source element. It is also possible to supply any extracted or input attribute value within the attribute definitions. Consider for instance the following SQL query to the Yahoo Travel Web site:

*Select Airline, Price from yahootravel where Departure=“BOS” and Destination=“SFO” and Month1=“5” and Day1= “19” and Month2= “6” and Day2= “1”*

When this query is executed, the input attribute values specified after the where clause replace the same name attributes enclosed between # signs in the post parameters as shown in Figure 2.4.

**5. Get, Post Methods & Authentication**

Caméléon spec files support both *get* and *post* methods when connecting to Web pages. A post example is shown in Figure 2.4. Method attribute of the post tag determines which method is to be used.

Most web pages perform authentication through forms. In connecting to those Web pages, get or post methods with parameter replacement can be used for authentication

purposes. In some other pages, however, the authentication is done through pop-up password windows. We handle these kinds of cases with the following scheme: (The username and password values have to be inputted within the SQL query, since they are coded as references in this spec file.)

```
<SOURCE URI=" http://game.etrade.com/cgi-bin/cgitrade/TransHistory ">
<AUTHENTICATION>
    <Realm>E*Trade Player (game)</Realm>
    <Username>#username#</Username>
    <Password>#password#</Password>
</AUTHENTICATION>
```

**6. Custom Cookies**

In Caméléon cookie handling is automatic as long as the cookies are set through headers. In some cases, cookies can be set in a non-standard way for example using Javascript API. To handle these cases we allow custom cookie setting as shown in example below (custom cookies used are: *jscript=1;path=/*)

```
<SOURCE URI="http://www.expedia.com/pub/agent.dll">
<COOKIE name="jscript">1</COOKIE>
<COOKIE name="path">/</COOKIE>
```

**7. JavaScript Interpretation**

JavaScript is used frequently in Web pages in creating the html document on the client side. In most cases, JavaScript does not pose a problem in wrapping Web pages, because it is usually used for cosmetic reasons. In some cases, however, not being able to interpret JavaScript may block the wrapper engine in getting to a desired page. One real example is the Expedia Web site, which requires interpreting JavaScript code and supplying the result as a post parameter. Caméléon spec files allow the interpretation of JavaScripts as shown in the following example.

```
<SOURCE URI="http://www.expedia.com/pub/agent.dll">
<JSCRIPT name="Time">
var d; d = new Date(); print(d.getTime());
</JSCRIPT>
```

In this example, the output of the JavaScript snippet is assigned to the Time attribute. We should note that the JavaScript code to be interpreted does not have to be static, and parameter replacement can be used in JScript tags as well.

**8. Prefix and Suffixes**

Figure 2.4 shows an example of prefix and suffix declarations. With these constructs it is possible to add static text before and after the extracted data values. The extraction engine returns the concatenation of prefix, pattern match and the suffix. By using parameter replacement feature, it also becomes possible to glue multiple extractions together.

**9. Delays**

Finally, we should mention another useful feature in spec file creation: delays. This is used when the wrapper engine requests data from a Web site, but has to wait a certain amount of time before getting an answer. We cover this case by specifying a delay element in the source declarations specifying the waiting time in terms of milliseconds. We show an example below:

<SOURCE URI="http://www.qixo.com/#Link#" DELAY="65000">

More details on writing spec files can be found in Caméléon user manual. [Firat et. al. 03].

## 2.3 IWrap: Instant Wrapper Generator

IWrap was our first effort in semi-automatic wrapper generation, with the objective of automatically creating spec files with some minimal user input. We designed a WYSIWYG interface as shown in Figure 2.5, allowing users to highlight sample data items and request auto generation of spec files. Please refer to the figure for the following operational description of IWrap.

In IWrap, the users first enter the URL of the page they want to wrap. The URL in most cases will include input parameters such as ticker symbols, search texts, etc. Page name and the input attributes are automatically inserted into the input table. Input table also contains the relation attribute, the method used in connecting to the page (GET or POST), which are expected from the user.

In the output table the user provides regional identifiers (i.e. BEGIN and END), and then highlights the text to be extracted with an associated identifier (i.e. the attribute name). It is also possible to highlight the extraction candidate using the source code pane when information to be extracted is hidden in an HTML tag.

When the user supplies all the information and clicks on the auto wrap button, the spec file creation begins with system messages explaining the progress in the messages window. If the creation is successful the user can immediately start issuing SQL queries to the wrapper engine.

In IWrap we did not adopt a learning algorithm, because we decided that annotating training data would be more costly than manual creation. Instead, we experimented with creating regular expression patterns from a single example page. Our initial results were promising for simple Web sites, but needed better success rates for more difficult ones. The operational details of IWrap with the algorithms used in generating the regular expressions are provided in [Firat et. al 1999].

## 2.4 Sample Applications

We wrapped numerous web sites using Caméléon and made some of them available online for demonstration. The samples can be reached from our demo Web site[13]. We have also developed demonstration applications that aggregate data from multiple web pages and present them in a unified interface. Illustrative examples include:

- Summarization and reorganization of seminar information by date from multiple separate departmental and local universities' online calendars
- Comparison of interest rates offered by various online Japanese banks
- Aggregation of personal financial information from all of your online banking, brokerage, and credit card accounts.
- Aggregation of air fare, hotel and car rental prices from popular online sources
- Aggregation of entertainment sources such as TV programs, events in a town, dating sites, etc.

[13] Currently at http://context2.mit.edu/

- Aggregation of educational sources such as online paper repositories and university web sites

Below, we give more detail on two of these applications.

### 2.4.1 Personal Investor Wizard

Personal Investor Wizard (PIW), aggregates data from ten different web sources including cnn, fox, yahoo, quicken, fortune and edgar. PIW continuously scrolls daily headlines (extracted from FoxNews and CNNfn), lets users view companies in a selected industry (obtained from yahoo), and display the competitors of selected companies (taken from quicken). A snapshot of this screen is shown in Figure 2.6. If the user wants to compare a set of companies, PIW displays in a second screen the profile info for each company (extracted from yahoo), the analyst recommendations (extracted from quicken), financial figures (extracted from edgar-online), and recent news (extracted from fortune). A snapshot of this screen is shown in Figure 2.7.

We built PIW using Java Server Pages (jsp) and simply embedded SQL queries in the jsp page. It is important to note that once the spec file for a web site is set up, it can be used by many applications and the developer of any of these applications merely views the web site as a traditional relational database. Development time of PIW, therefore, was quite short.

Because Caméléon accepts SQL queries and has a Java Servlet version, it is also very easy to call it from other applications. The Java version of PIW, an example of this flexibility, is also available in our web site for download

### 2.4.2 Airfare Aggregation

Airfare aggregation application displays price and airline information from nine different airline and aggregator sites given departure and destination locations and dates. We show snapshots from this application in Figure 2.8. This application was built very easily by embedding SQL queries in an ASP.NET application. These SQL queries were sent to the Caméléon wrapper engine as if it was a relational database system, which returned results as data sets.

In addition, the airfare application can send complex queries against the relational front end, to search intervals rather than fixed dates. The ability of a wrapper engine to handle complex queries with its relational capabilities frees the application developer from having to go through the planning

## 2.5 Discussion & Conclusion

Caméléon wrapper engine is unique in combining data extraction with traditional database techniques, thus allowing easy interoperability between semi-structured and structured data sources. The core wrapper engine is able to provide a robust infrastructure for web automation as defined in [Allen 97]. Specifically:

- It has full interaction with HTML forms, i.e. it supports both get and post methods;
- It handles both HTTP Authentication and Cookies;
- Both on-demand and scheduled extraction of targeted web data are possible as demonstrated by PIW Java version;
- It facilitates aggregation of data from a number of web sources;

- It can extract data across multiple web sites through chaining;
- It is very easy to integrate with traditional application development languages and environments as it provides a SQL interface to web pages.
- Our declarative way of specifying extraction rules provides a clean framework for managing change in both the locations and structures of Web documents.

It can, however, be improved in a number of ways with additional research. Some of the areas that need more work are:

- Increasing the expressiveness of spec files (e.g. handling non-deterministic multiple page traversals)
- Improving the semi-automatic wrapper creation work with learning approaches using minimal examples.
- Creating monitoring tools, that would auto update spec files if possible, or at least detect the need to update spec files.

From here on, in this Thesis, we will assume that web sources can be viewed as databases through the use of data extraction technologies such as Caméléon, and focus on the problems related to data interpretation in the coming chapters.

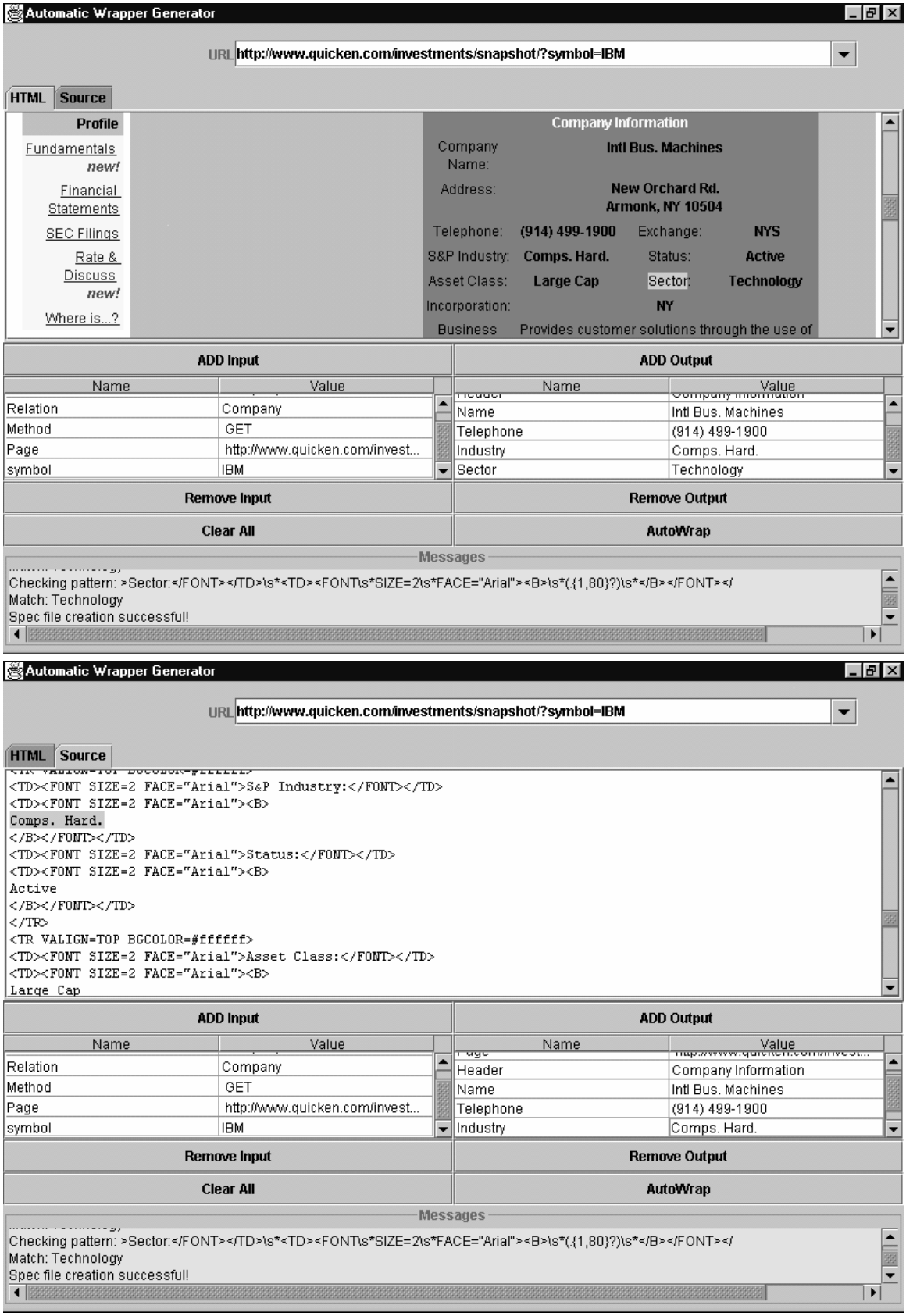



Figure 2.5. IWrap Snapshots

.

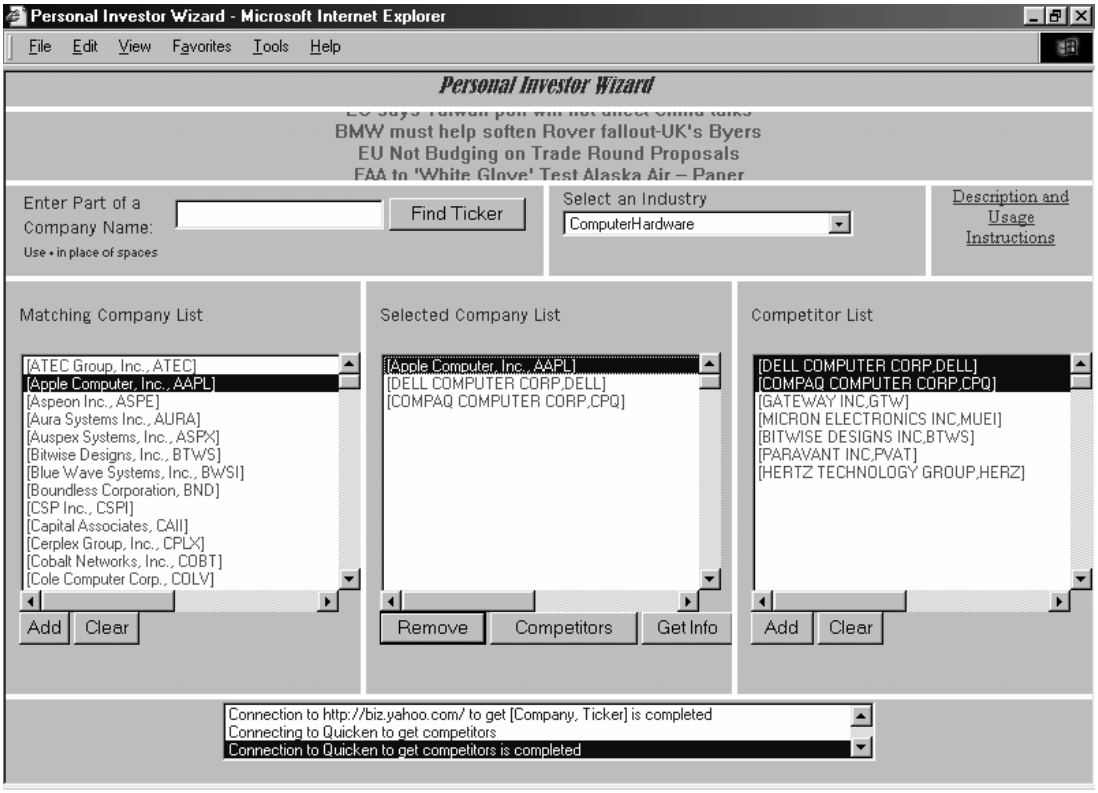



Figure 2.6. Personal Investor Wizard - Main Screen

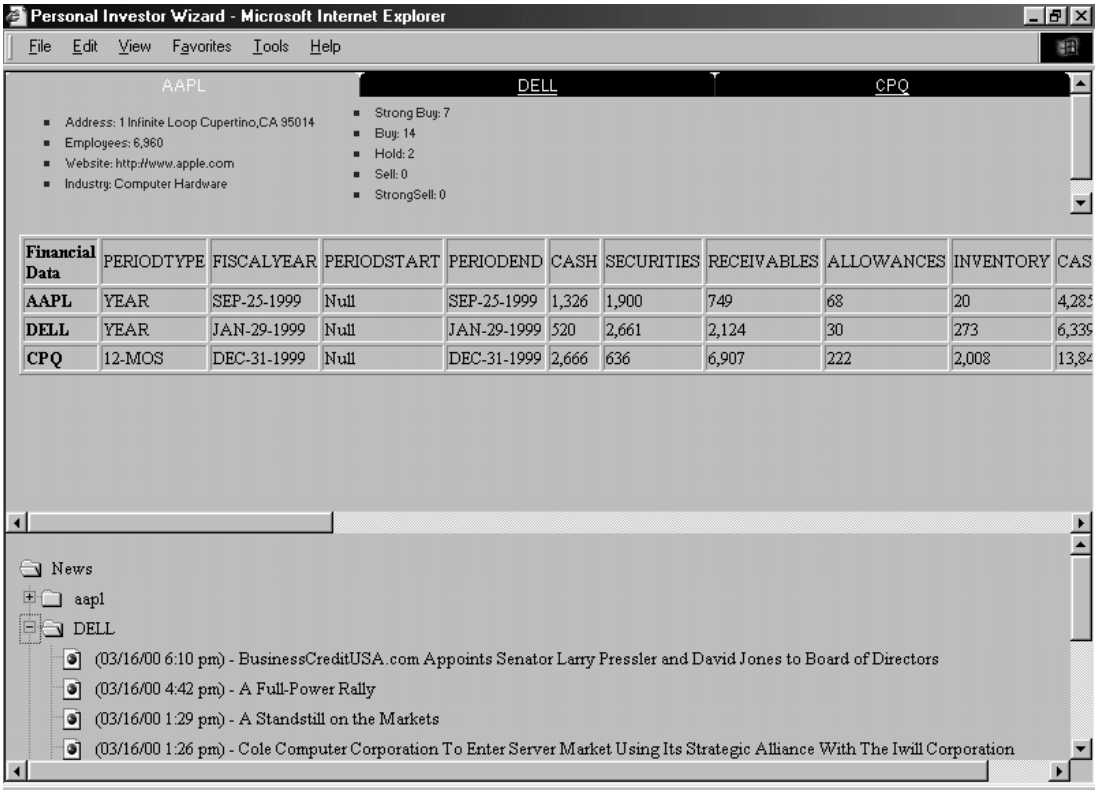
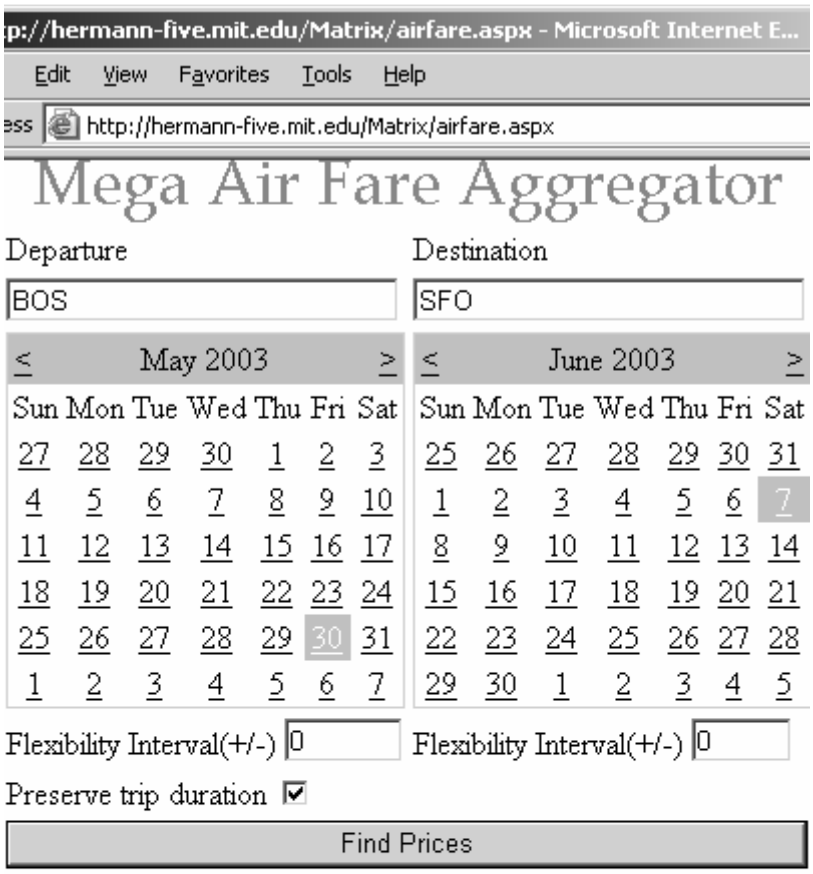



Figure 2.7. Personal Investor Wizard - Comparison Screen



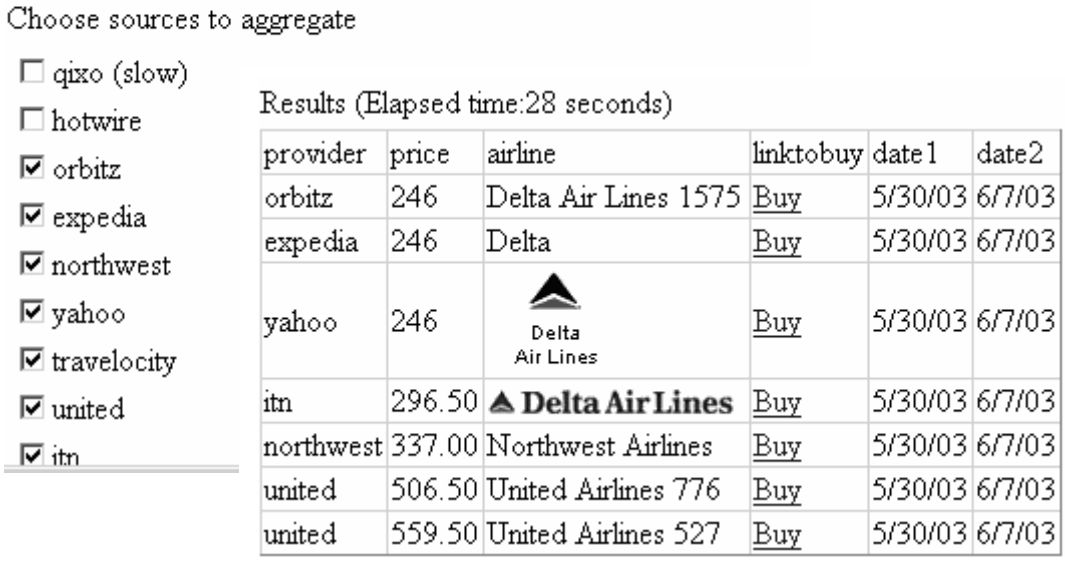




Figure 2.8 Air Fare Aggregation Screen Shots

"England and America are two countries separated by the same language."

George Bernard Shaw.

# Chapter 3

# Data Interpretation

In the previous chapter, we focused on data extraction, an effort to reconcile *technical* and *interface heterogeneities* [Busse et. al 99] among information systems. Many information systems cannot communicate meaningfully, however, even when these *syntactical heterogeneities* are eliminated. Real challenges of achieving interoperability among autonomous and heterogeneous systems lie in dealing with issues related with interpretation of data.

Data interpretation involves combining data with contextual knowledge that collectively determine the frame of meaning for that data. Consider, for example, phone numbers, which are often exchanged without the area code, and almost always without the country code. Without the contextual knowledge which determines the area codes from spatial information, the frame of meaning for phone numbers may be too broad to be useful (it may belong to hundreds of users in different area codes), and even worse, may be misunderstood (could be taken as a number within the interpreter's area) if contextual borders are trespassed.

When data originating from different contexts are brought together, many heterogeneities are observed. In this chapter, we provide a classification of semantic heterogeneities we observed in a financial case study in which we examined data collected from various sources. Our primary objective in this chapter is to familiarize the reader with existing approaches to dealing with these heterogeneities in information integration, which will hopefully lead to a better understanding of the subtleties of our approach described in the following chapters.

## 3.1 Dimensions of Semantic Heterogeneity

In our past information integration research projects, we were often puzzled by seemingly contradictory data within one database or across multiple databases. In one of these projects, we examined *Primark*'s *Worldscope*, *DataStream* and *Disclosure* databases and data definition manuals as well as *Security Exchange Commission* (SEC) *Company*

*Filings* and several other web-based financial sources[14] to have a deeper understanding of the reasons behind semantic heterogeneities.

We compared *"Net Sales", "Net Income", "Total Assets", "Number of Employees", and "Five-Year Growth in Earnings per Share"* accounting data items for a given company across these data sources and found significant variations. In Table 3.1, variations between Disclosure and Worldscope databases range from 4 to 92 percent for these five accounting data items for the same set of companies.

| ACCOUNTING DATA ITEMS | % OF VARIATIONS |
|---|---|
| Net Sales | 20 |
| Net Income | 20 |
| Total Assets | 4 |
| Number of Employees | 40 |
| Five-Year Earnings Growth per Share | 92 |

**Table 3.1.** Variations between Disclosure and Worldscope databases

We reviewed our findings with Primark representatives to discover that variations could be attributed to different reporting standards, namely data item definitions and representations, used by different databases. Different types of users prefer to view company financial data in different ways depending on their job functions as illustrated in Figure 3.1.

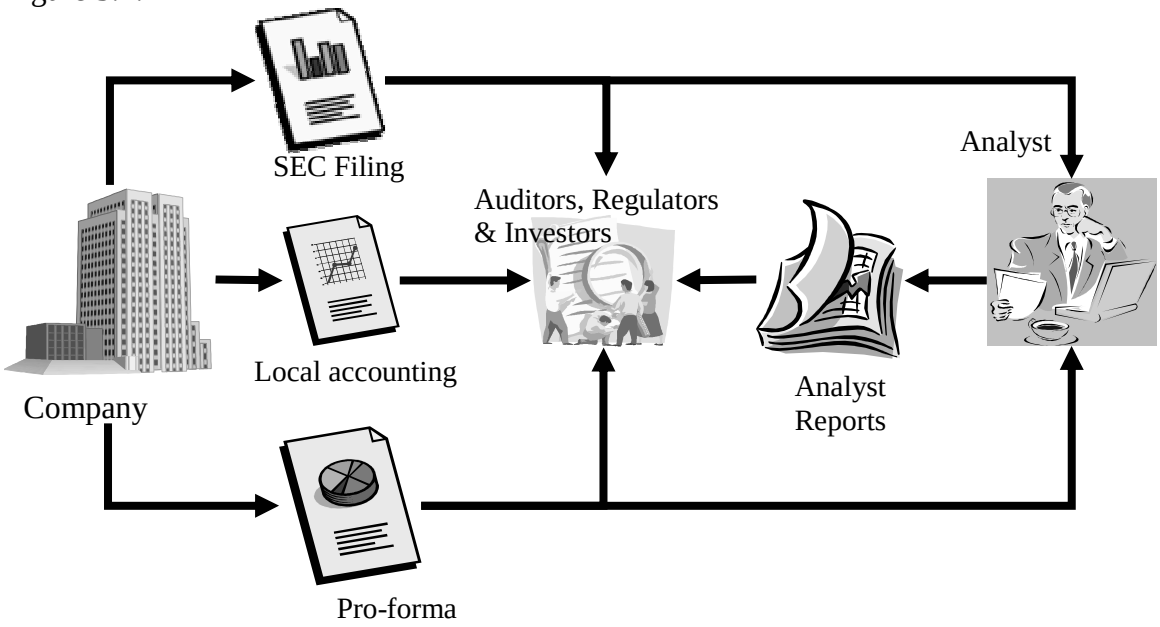



Figure 3.1 Different people need different forms of data

[14] Including Hoovers, Yahoo, Market Guide, Money Central, and Corporate Information

.

As seen in the figure, a company may provide data to public through:

- *official filings* (e.g. to Security Exchange Commission (SEC) using Generally Accepted Accounting Principles (GAAP));
- *pro-forma press releases* as a management interpretation of financial results; or
- the use of *local accounting principles* (e.g. using UK GAAP)

Furthermore analysts may process these statements and release processed and aggregated data in yet other forms to allow for meaningful performance analysis.

When we tried to integrate data sources adopting different views of data, we noticed several semantic heterogeneities. Below, we elaborate on three dimensions of semantic heterogeneity: *contextual, ontological and temporal*. The relationship between these heterogeneity types are illustrated in Figures 3.5 and 3.6, and explained in the following subsections

Before going into the details we should briefly explain what we mean by the terms we will frequently use in the next sub sections: primarily, the terms *intensional* and *extensional*. By *intensional* we refer to *abstract descriptions* which identify concepts without enumerating its members. *Extensional* is the antonym of intensional, and refers to *enumerative descriptions* of concepts with its physical members. For example, as shown in Figure 3.2, the physical collection of records in a database relation is known as the extensional relation, where as the schema of a relation is known as the intensional relation. We will delay the formal definition of ontology to Chapter 5, and suffice it to say that an ontology is a collection of intensional descriptions.

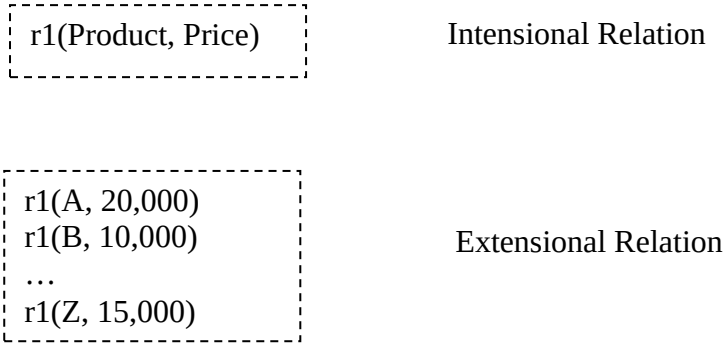



Figure 3.2 Intensional vs. Extensional Relations

## 3.1.1 Contextual Heterogeneity

Often times, an intensional description is not specific enough to determine the exact form of its extension. Consider for instance the following intensional description of a concept called "price": "*the amount as of money, asked for or given exchange for something else without the inclusion of tax*". This definition leaves out "price" attributes such as currency, and scale allowing disparate adoptions of currency and scale for price entities in data sources and receivers. This is illustrated in Figure 3.3, with the multiple

mappings of the intensional relation r1(Product, Price) to extensional relations with different currencies and scale factors.

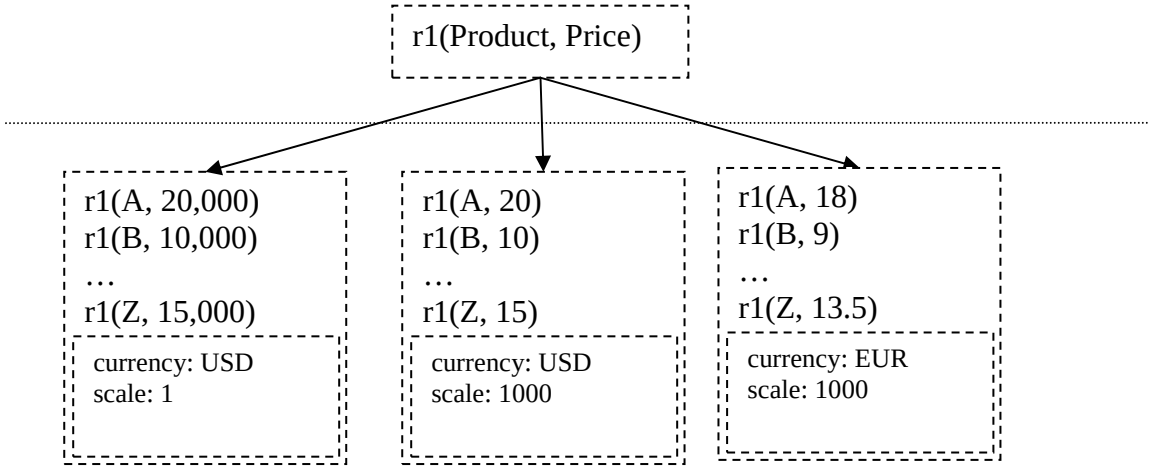



Figure 3.3 Multiple Extensions of an Intensional Relation

These are heterogeneities that [Batini et al. 86] refers to as the two or more not identical representations of the same concept. In information systems, we observe this type of variations when entity type definitions corresponding to the same real world entity are flexible enough to allow data sources and/or receivers choose their own representation. We show an example in Figure 3.4, in which the sales numbers of FIAT, an Italian motor company, are represented differently in Worldscope and Market Guide data sources.

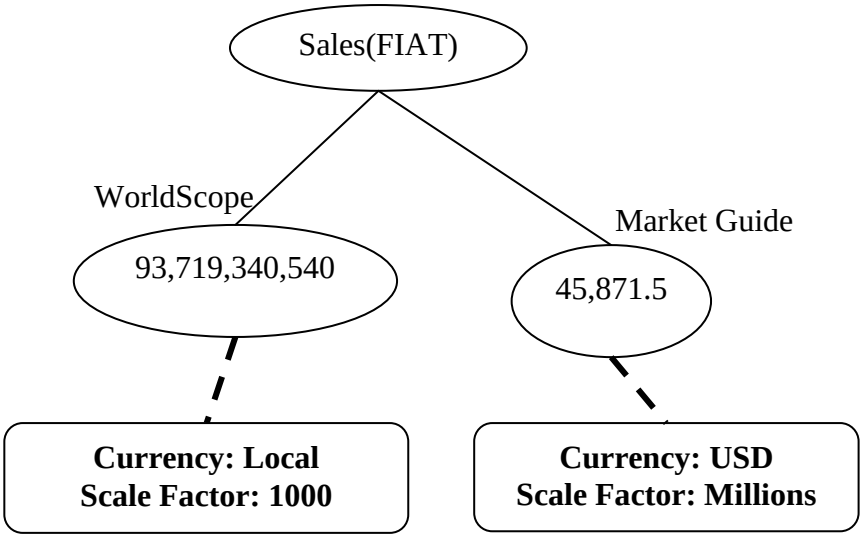



Figure 3.4 Contextual Heterogeneity in Worldscope and Market Guide Data Sources

## 3.1.2 Ontological Heterogeneity

Ontological heterogeneity is the heterogeneity in the intensional description of concepts that are somewhat related. For example, if we define another price concept, "price (+tax)" as *"the amount as of money, asked for or given in exchange for something else <u>with</u> the inclusion of tax"*, this would constitute an ontological heterogeneity with the concept "price (nominal)" described in the previous section because there is a definitional conflicts concerning the inclusion or exclusion of tax in the price. As shown in Figure 3.5, the price amounts *21,000 USD* and *17K EUR* exhibit ontological heterogeneity because they belong to extensions of different intensional descriptions.

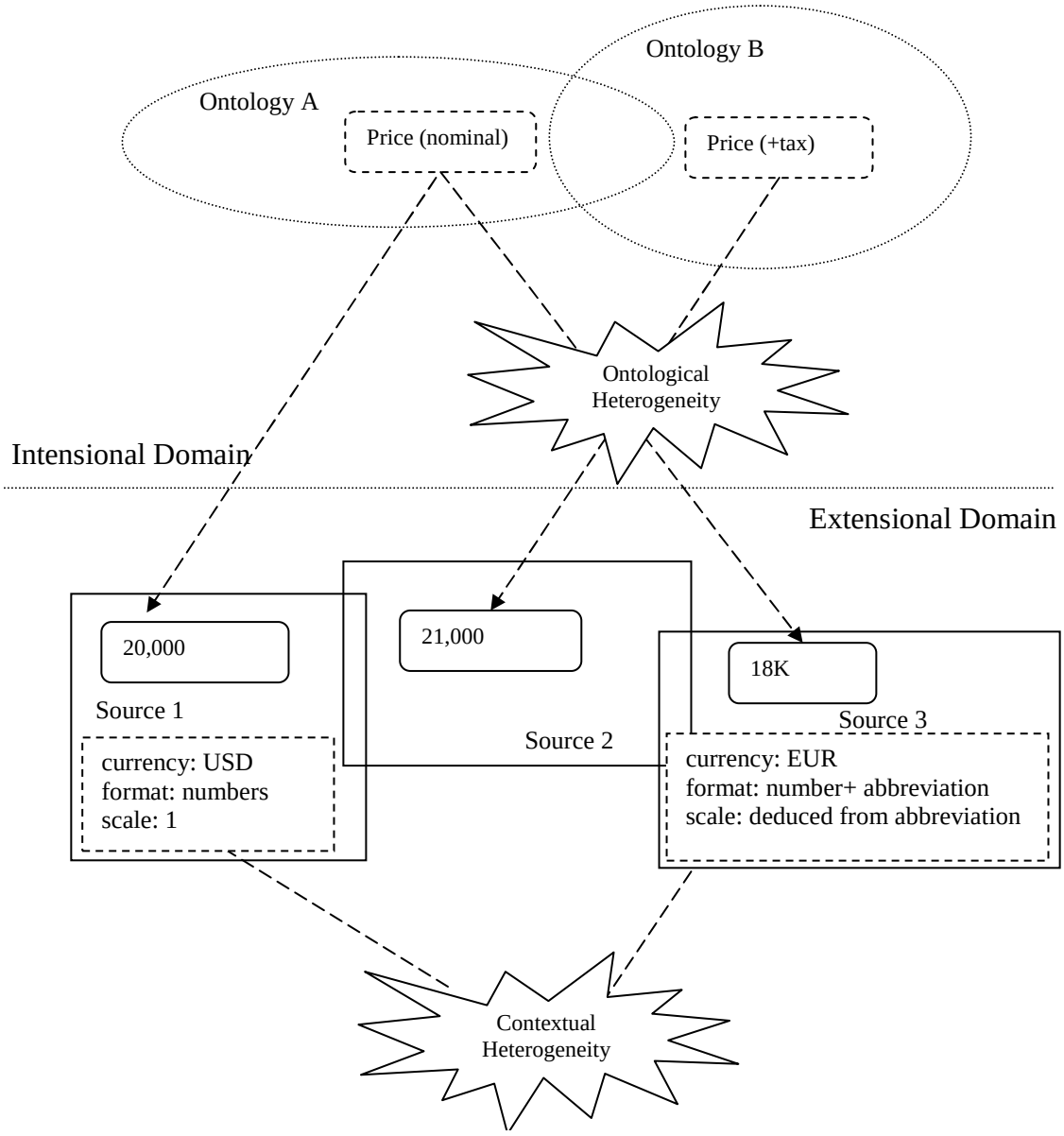



Figure 3.5 Contextual and Ontological Heterogeneities

In information systems, we observe this type of heterogeneity, when databases differ on entity type definitions. For example, the majority of definitional variations in financial

information systems could be attributed to the inclusion or exclusion of various accounting items such as *"Depreciation and Amortization", "Excise Taxes", "Earnings from Equity Interests", and "Other Revenue"* from the financial data items. Similarly, variations in "*Total number of Employees"* could be attributed to inclusion or exclusion of "*Temporary Employees"*, "*Employees of Subsidiaries"* as well as the time of reporting. In addition, some of the variations in "*5-Year Earnings Growth per Share"* numbers could be attributed to the lack of accounting for fluctuations in foreign currency.

### Equational Ontological Conflicts (EOC)

Despite having differing definitions, entities can usually be related to each other when one or more entities uniquely determine the value of one or more other entities. For example, for certain companies, the *"Pretax Income"* can be derived from *"Pretax Profit"* and *"Assoc. Pretax Profit"* attributes in another, as shown below:

**"Worldscope. Pretax Income"= "Datastream. Pre-tax Profit" – "Datastream. Assoc. Pre-tax Profit"**

We label the heterogeneity in the way data items are *calculated* from other data items *in terms of definitional equations* as equational ontological conflicts. We show more examples of EOC in Table 3.2 below.

| Source A | Source B |
|---|---|
| # of customers = # of end_customers + # of distributors | # of customers = # of end_customers + # of prospective customers |
| Profit = Net Sales – Cost of Goods | Profit = Net Sales – Cost of Goods – Depreciation |
| P/E Ratio = Price / Earnings(last 4 Qtr) | P/E Ratio = Price/ [Earnings(last 3 Qtr) +Earnings(next quarter)] |
| Price = Nominal Price + Shipping | Price = Nominal Price + Shipping + Tax |

Table 3.2 Example Equational Ontological Conflicts

## 3.1.3 Temporal Heterogeneity

As shown in Figure 3.6, temporal heterogeneities are orthogonal to both contextual and ontological heterogeneities and arise because of changes in the intensional or extensional descriptions of concepts over time. In Figure 3.6, Ontology B shifts from one intensional definition of price to another, and the extensional object represented by the *21,000 USD* shifts to a different currency, scale factor and format by becoming *17K EUR*. The first shift is the combination of ontological and temporal heterogeneities,

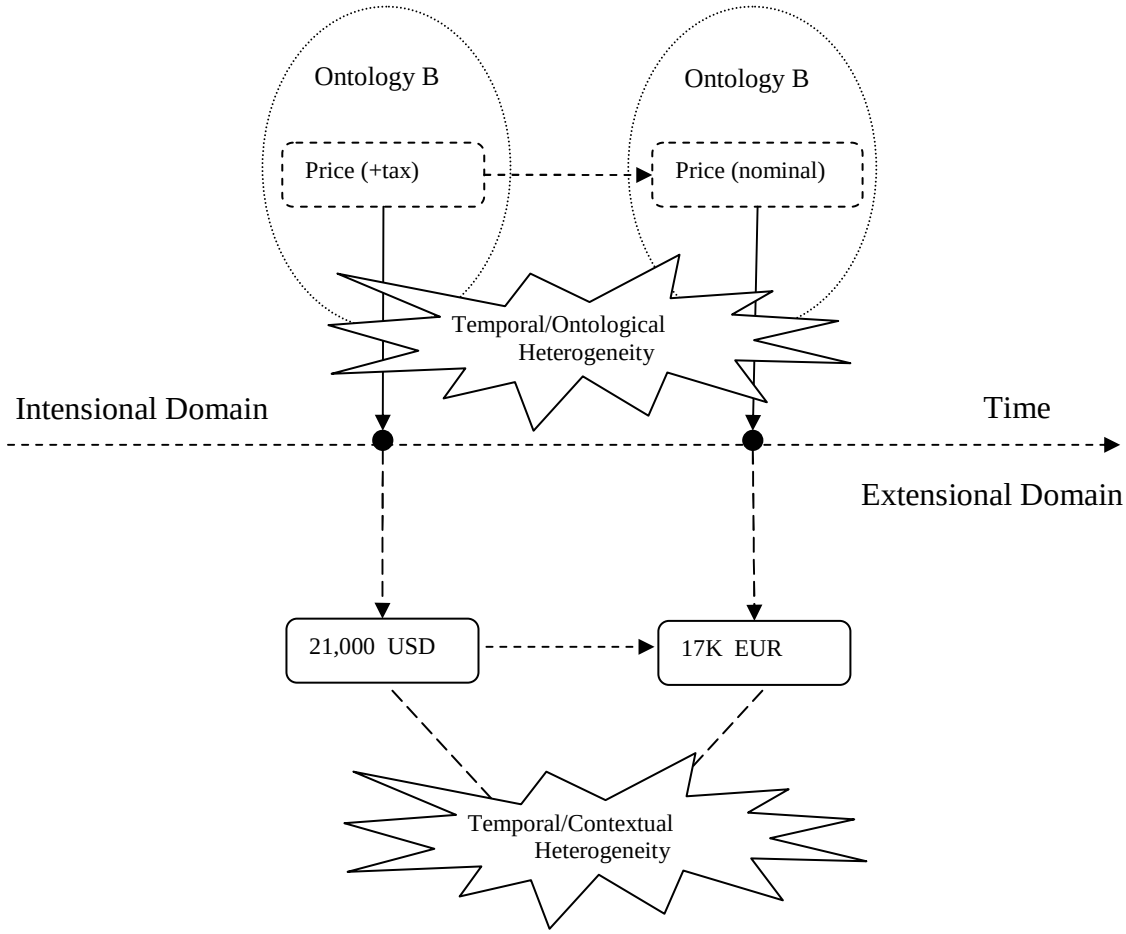



Figure 3.6 Temporal Heterogeneity

whereas the second one is a combination of contextual and temporal heterogeneities.

In information systems, temporal variations arise when entity values or definitions belong to periods that exhibit contextual or ontological heterogeneities. Definitions of data terms, for example, may change over time as seen in the example below. The three-way dependency between the Worldscope, Disclosure, and SEC databases for Exxon is different before and after 1996.

*For Exxon after 1996:*
*"Worldscope. Revenues" = "Disclosure. Net Sales" –" SEC. Earnings from Equity Interests and Other Revenue" –" SEC. Excise Taxes"*

*For Exxon before 1996:*
*"Worldscope. Revenues" = "Disclosure. Net Sales" – "SEC. Excise Taxes"*

Temporal heterogeneity should not be mixed with contextual or ontological heterogeneities involving temporal concepts across information sources. For example, two sources reporting financial numbers quarterly vs. annually may have contextual heterogeneity, but a single source shifting from quarterly to annual reporting at a certain year is said to have a temporal heterogeneity. Temporal heterogeneity is about temporal changes in the intensional or extensional descriptions of data sources.

.

### 3.1.4 On the Relationship between Contextual and Ontological Heterogeneities

The primary distinction between ontological and contextual heterogeneities is that the former refers to heterogeneity in *explicit knowledge* (i.e. ontological definitions), whereas the latter refers to heterogeneity in *implicit knowledge* (i.e. contextual knowledge). This distinction between ontological and contextual heterogeneities is also a mode of connection between them, allowing us to transform one to another by making knowledge implicit or explicit.

If we could make everything explicit in an ontology --which may be neither possible nor desirable-- by defining every term with uttermost detail (e.g. price in USD with a scale factor of 1 including tax…), there would not remain anything implicit about data once they are mapped to the ontological terms. In such a case, all conflicts would be ontological in nature and no contextual heterogeneity would exist (see Figure 3.7)

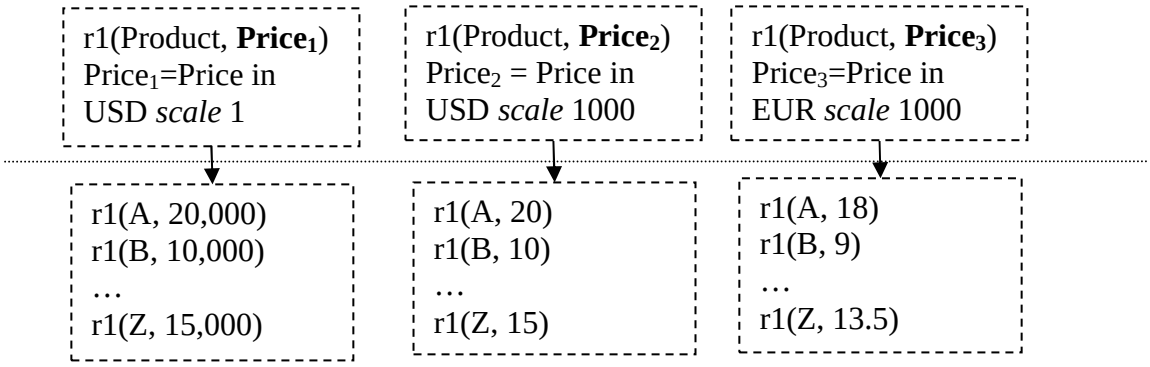



Figure 3.7 Extreme Example for Ontological Heterogeneity

The other extreme would be mapping all data to "*thing*" (i.e. the most basic term in an ontology) in the ontology and treating the rest as contextual knowledge. In such a case, we could only talk about contextual heterogeneity, since no definitional heterogeneity exists at the ontological level.

When we integrate data sources with their respective ontologies (we will accept schemas as ontologies here) by using a shared ontology, we can decide how much to make explicit in the ontology. Take for example *price (nominal)* and *price(+tax)* ontological heterogeneity between two data sources. This heterogeneity may be treated as a contextual heterogeneity, by adopting a more general definition of price that subsumes both concepts, and by leaving their difference to be articulated as part of the contextual knowledge. In fact, this is the approach we take in this Thesis: we treat ontological heterogeneities as contextual heterogeneities by using relatively generic knowledge in the ontologies, and relatively particular knowledge in contexts.

.

### 3.1.4 Related Work

There have been several attempts in the literature to classify data heterogeneities [Kim & Seo 91, Kashyap & Sheth 96, Goh 97, Busse et. al 99]. Almost all of these attempts are primarily from the database perspective, as evidenced by major focus on schematic conflicts. Kashyap and Sheth, for example, classify everything under schematic conflicts (corresponding to ontological heterogeneity in our classification), and view conflicts such as scaling and units as schematic conflicts pertaining to domain definitions. Kim and Seo have the additional category of data conflicts, in which they draw attention to the heterogeneity in the quality of data (e.g. incorrect, obsolete data) as well as different representations of the same data (e.g. unit and scale differences), which corresponds to contextual heterogeneities in our framework. In [Goh 97], Goh talks about naming, scaling & units, and confounding conflicts. Confounding conflicts, which is particularly interesting, refers to those arising from the confounding of concepts, which are in actual fact distinct (e.g. latest trade price as of now vs. latest trade price with a 20 minute delay). We would categorize confounding conflicts under ontological conflicts in our framework.

It is beyond the scope of this Thesis to come up with a synthesis of all semantic heterogeneities in great detail. Our classification of semantic heterogeneities in section 3.1 is based on a practical concern that affects the design of our integration framework. We knew, for instance, how to handle contextual heterogeneities using the COIN framework [Goh 97], but did not have a way to handle ontological heterogeneities. In this Thesis, we transform ontological heterogeneities into contextual heterogeneities as described in the preceding sub section and use an extended version of the COIN framework, ECOIN, to reason with them. Handling temporal heterogeneities will require further enhancements and is left as future work.

## 3.2 Major Approaches to Achieving Interoperability

Over the last two decades there have been several studies on database integration under a variety of titles such as multidatabase systems, heterogeneous database systems, and federated information systems [Busse et. al 99]. These approaches have been grouped in the literature as static vs. dynamic [Kuhn et. al 91], global vs. local schema [Litwin and Abdellatif 87], and tightly vs. loosely coupled [Goh 97, Arens and Knoblock 96] approaches. These groupings can roughly be thought of referring to the same distinction characterized in [Goh 97] by:

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

In the following subsections, we analyze these approaches under the headings of tightly and loosely coupled approaches with the exception of our predecessor system COIN, which adopts a unique, and in some ways a hybrid approach. In Table 3.3, we provide a grouping of some of the existing prototype systems according to this criterion [partially adopted from Goh 97].

.

| | Tightly Coupled Systems | Loosely-coupled Systems |
|---|---|---|
| **Logic-based Data Model** | Information Manifold [Levy 98]<br>InfoMaster [Duschka and Genesereth 97]<br>HERMES [Subrahmanian et al. 00]<br>Carnot [Collet et al. 91] | VIP-MDBMS [Kuhn and Ludwig 88] |
| **Object-Oriented Data Model** | DISCO [Tomasic et. al 98]<br>SIMS [Arens and Knoblock 96]<br>Pegasus [Ahmed et al. 91] | TSIMMIS [Garcia-Molina et al. 95]<br>O*SQL [Litwin 92] |
| **Functional Data Model** | Multibase [Landers and Rosenberg 82] | |
| **Relational Data Model** | ADDS [Breitbart and Tieman 84]<br>Mermaid [Templeton et al. 87]<br>Garlic [Carey et. al 95] | MRDSM [Litwin and Abdellatif 87] |

Table 3.3 A Categorization of Existing Prototypes

### 3.2.1 Tightly Coupled Approaches

In tightly coupled approaches, the objective is to insulate the users from data heterogeneity by providing a unified view of the data sources, and letting them formulate their queries using that global view. A system administrator takes the task of creating a global schema before the system can be used. In *bottom up* approaches the global schema is constructed out of heterogeneous local schemas by going through the tedious process of schema integration [Batini et al. 86]. In *top-down* approaches global schema is constructed primarily by considering the requirements of a domain, before corresponding sources are sought.

Virtually all data integration systems in this category can be viewed as a triple (G, S, M) [Lenzerini 03], where G is the global schema, S is the source set, and M is the mapping between G and S as illustrated in Figure 3.8 The primary challenge of these integration systems is to rewrite a user query expressed over the global schema (q(A',B') in the figure), in terms of queries spanning the source set (q(A) & q(B) in the figure) by using the mappings between them. Mappings in these systems can be expressed in two ways: local as view (LAV) or global as view (GAV). In LAV each source is described through the global schema (A ← A'), whereas in GAV, global schema is expressed using the sources (A' ← A). A concrete example is given in Table 3.4. In this example, the global schema consists of three predicates {car(C), price(C,P), model(C,M)} and there are two sources {s1(C,P), s2(C,M)} reporting the prices and models of cars respectively. The problem of rewriting queries posed on the global schema in terms of the local sources has been coined as *answering queries using views* in the literature [Halevy 00], and studied extensively [Pottinger and Levy 01]. Research in this area focuses more on the mechanics of answering queries using views problem than identifying and conquering challenging semantic issues.

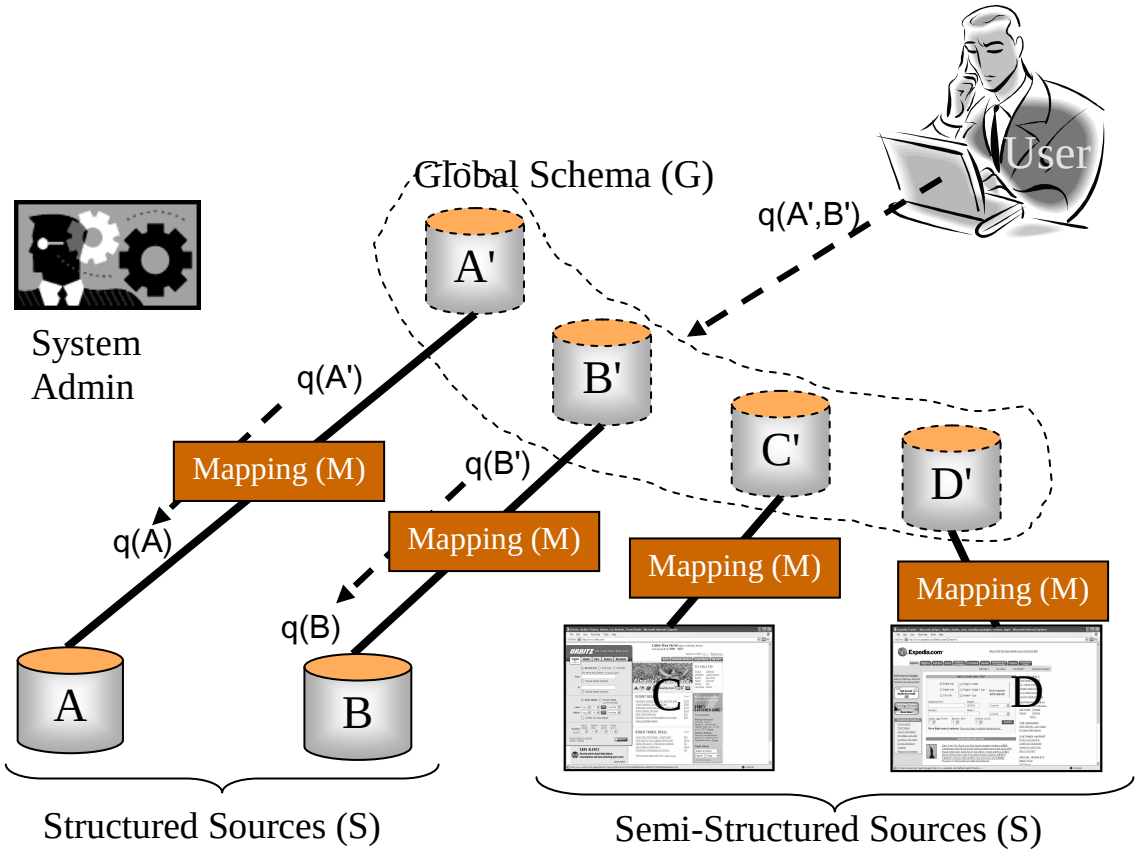



Figure 3.8 Tightly Coupled Approaches

| Global As View | Local As View |
|---|---|
| car(C) ← s1(C).<br>car(C) ← s2(C).<br>price(C,P) ← s1(C,P).<br>model(C,M) ← s2(C,M). | s1(C, P) ← car(C), price(C, P).<br>s2(C, M) ← car(C), model(C,M). |

Table 3.4 GAV vs. LAV

In tightly coupled approaches, data heterogeneities between sources are resolved by mapping conflicting data items to a common view. In the Pegasus system, for example, supertypes and functions are used for this purpose. To circumvent a conflict between two data types a supertype, and a function that acts on the instances of the supertype is created. The function provides a mapping between the data in conflicting sources and the canonical form adopted by the global schema. An ontological heterogeneity example in which two data sources have conflicting price definitions is shown below:

.

```
CREATE SUPERTYPE trip of trip1, trip2;
CREATE FUNCTION price (trip x) →
Real r AS
IF trip1(x) THEN airfare(x) + servicecost(x) + shippingcost(x)
ELSE if trip2(x) THEN airfare(x) + tax(x)
ELSE ERROR;
```

In this example, the *trip* supertype subsumes trip1 and trip2 types, and imposes a standard price definition by using a function to map conflicting data values to this standard definition.

## 3.2.2 Loosely Coupled Approaches

Loosely coupled approaches object to the feasibility of creating unified views on the grounds that building and maintaining a global schema would be too costly. Instead they aim to provide users with tools and extended query languages to resolve conflicts themselves as illustrated in Figure 3.9. The approach relies on the assumption that data with the same meaning usually have the same or similar names, which can be identified by the users easily [Kuhn et al. 91]. Furthermore, the user is assumed to understand and resolve conflicts with the provided tools. One of the best representatives of this approach is the MRDSM system that introduces the concept of *dynamic attributes* to deal with conflicts between data sources [Litwin and Abdellatif 87]. A dynamic attribute is a virtual column with a dynamically assigned value by using arithmetic operators, functions and/or queries. The example we have given in the tightly coupled case, would be expressed in the loosely coupled case as follows (assuming that the data sources are Orbitz and Travelocity and the user wants to see the final prices (including taxes, service and shipping charges) of tickets from both databases):

```
USE (orbitz o) (travelocity t)
D-COLUMN HOLD (t.price)
t.price = t.airfare + t.shippingcost + t.servicecost
D-COLUMN HOLD (o.price)
o.price = o.airfare + o.tax
SELECT airline, price FROM orbitz WHERE Destination= “BOS” and
Arrival= “IST” and Ddate= “8/1/03” and Adate = “9/1/03”
UNION
SELECT airline, price FROM travelocity WHERE Destination= “BOS” and
Arrival = “IST” and Ddate= “8/1/03” and Adate = “9/1/03”
```

In the above MSQL (a variant of SQL used in MRDSM) D-COLUMN declaration denotes dynamic columns followed by the arithmetic expression that defines the value of the virtual column. Later, these virtual columns can be remembered and used in the

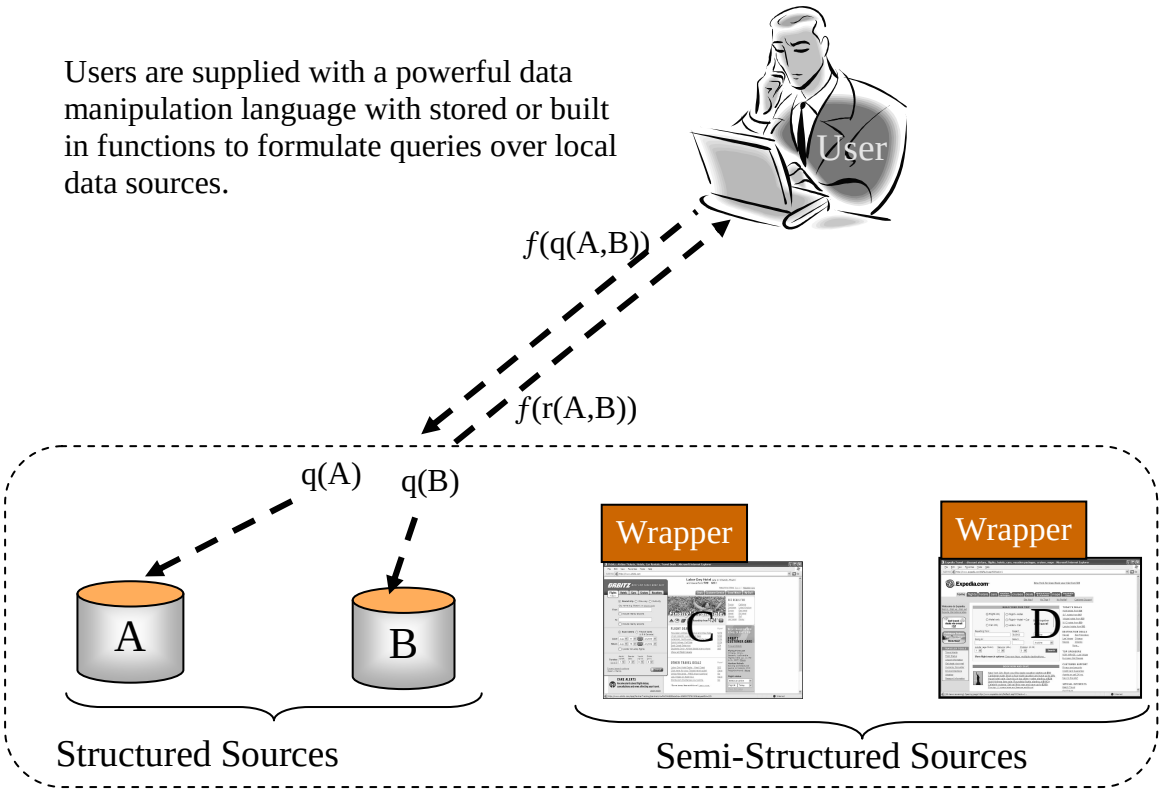



Figure 3.9 Loosely Coupled Approaches

standard SQL statements. Thus, when the SQL query against Travelocity (t) is executed the system automatically adjusts the value of price according to the pre-specified formula. Note that airfare definitions between Travelocity and Orbitz are different, as the latter includes shipping and service costs but excludes the tax.

As opposed to the tight coupling approach, loose coupling allows users to get the data in more than one view with the provision of appropriate mappings. This makes the loosely-coupled approaches more flexible compared to tightly-coupled ones. Unlike the tightly coupled approaches, however, the user, not the system administrator, has the burden of writing mapping functions in every query constructed.

One of the interesting aspects of MSQL is that it attempts to apply a limited form of symbolic and numeric manipulation techniques to automatically invert and simplify equations specified in dynamic attribute declarations [Litwin and Vigier 87]. For example, a dynamic column could contain the following formula:

$$2*balance^2 + 2*balance - 24 = 0$$

and the system would determine that the balance is 4 or -5 by making calls to Macsyma equation solver with the above equation. (A method for choosing one of the solutions is explained in [Litwin and Vigier 87]) In Chapter 6, we explain how this approach compares with the approach we adopted in solving *equational ontological conflicts*.

.

### 3.2.3 Short-comings of Existing Approaches

While both tightly-coupled and loosely-coupled approaches offer solutions to semantic interoperability among autonomous and heterogeneous data sources, they also pose significant problems.

The fundamental problem with tightly coupled approaches is that it lacks flexibility in providing multiple views of data sources. As we have discussed early in this chapter, different people may want to view data sources in different ways. In tightly coupled approaches user views can only be constructed manually by a system administrator. In loosely coupled approaches, users can construct their own views, making the system more flexible and responsive, but they have to understand the conflicts between their views and those of data sources. In our ECOIN approach, we combine the best of these two worlds, and allow the automatic construction of multiple views with the help of declarative data semantics and a mediator that automatically detects and reconciles conflicts between data sources and receivers. Compared to loosely-coupled approaches, our approach lessens the burden of the user.

In tightly-coupled systems the detection and reconciliation of data conflicts are not optionally visible to the user. Semantic conversions are buried inside the wrappers and not easily inspectable. Loosely-coupled approaches provide some level of transparency as the users define the semantic conversions themselves, but there is no system service that optionally provides a list of the detected and resolved conflicts between the data sources and user views. In ECOIN, conflict detection and resolution is optionally visible to the user with the provision of an intensional answer in addition to an extensional answer. This will be explained in more detail in the coming chapters.

Finally, in adopting a tightly coupled approach, system developers face the complexities of building and maintaining a global schema. Tightly coupled approaches suffer from the scalability problem, as it becomes more and more difficult to maintain a schema with large number of sources.

ECOIN combines the best aspects of both tightly and loosely coupled approaches and provides a hybrid approach. In the coming chapters we explain the details of our approach.

.

# Chapter 4

# Context Interchange Strategy

In the previous chapter we explained various approaches to achieving semantic interoperability among heterogeneous information systems. In this chapter, we describe the Context Interchange (COIN) strategy, which is the foundation of our approach to achieving semantic interoperability among autonomous and heterogeneous systems.

COIN *strategy*, first articulated in [Siegel and Madnick 91, Sciore et al. 94], and later founded on a formal conceptual basis in [Goh 97] has the basic tenets that

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

These novel ideas signify an important departure from the existing tightly and loosely coupled approaches to semantic interoperability. Unlike tightly coupled systems (e.g. Pegasus [Ahmed et al. 91]), COIN strategy does not burden system administrators with *a priori* detection and reconciliation of semantic conflicts, but only requires the creation of a shared ontology that enables the declaration of conflicting data semantics; and the provision of conversion functions that will be used when these conflicts are automatically detected.

COIN strategy also differs from the loosely coupled systems (e.g. MRDSM [Litwin and Abdellatif 87]) in which the user is responsible for identifying and resolving conflicts before issuing queries. This responsibility is shifted to the *context mediator* in the COIN strategy. Instead, the receivers (i.e. users) and data sources are assigned the lesser responsibility of furnishing a declarative specification of how they interpret data.

COIN strategy was materialized through the description of a data model, reasoning algorithm and a prototype (collectively called COIN) in [Goh 97]. Both the data model and the reasoning algorithm were, however, silent on several aspects of the COIN strategy. In particular, the case of dealing with ontological heterogeneities was not thoroughly examined. Our work in this Thesis provides an extended formal re-

conceptualization of COIN strategy (ECOIN) which improves the COIN data model, reasoning algorithm, and system prototype.

We start this chapter with two examples that illustrate the features of COIN strategy from the user perspective. We, then, continue to explain the structural elements of COIN from the system perspective, and also provide an overview of the reasoning algorithm. Finally, we compare ECOIN with COIN, and outline the new features introduced by our work that paves the way for a more detailed account of ECOIN in following chapters.

# 4.1 COIN Strategy by Example

## 4.1.1 Air Fare Scenario

In Chapter 2, we mentioned that with the use of Caméléon wrapper engine airfare providers on the Web could be treated as if they were databases. Consider now the slightly dramatized scenario shown in Figure 4.1. A price sensitive Turkish student is looking for a round trip airfare from Boston to Istanbul, first leg on June 1st and second on August 8th 2003 from Yahoo travel site. The contexts of the data sources and the user (i.e. the way they interpret data) are shown in the figure. First the user wants to know which airlines are available for his trip and formulates his SQL query as follows:

```
Q1:   SELECT Airline FROM Yahoo
      WHERE DepDate = "01/06/03" and ArrDate=  "01/08/03"
      and  DepCity= "Boston" and ArrCity= "Istanbul";
```

Without any mediation, this query would return an empty answer, because Yahoo expects city codes instead of city names and dates in American format. If this query was submitted to COIN, however, the query would be rewritten into the following mediated query:

```
MQ1:  SELECT Airline
      FROM yahoo,
      (select Airport from cityAirport where city= "Boston") depCode,
      (select Airport from cityAirport where city= "Istanbul") arrCode,
      WHERE DepDate = "06/01/03" and ArrDate="08/01/03" and
      DepCity= depCode.Airport and ArrCity= arrCode.Airport;
```

and the system would return the following result set:

| **Airline** |
|---|
| British Airways |
| Lufthansa |

## Context of Yahoo

Price means *one way nominal price*

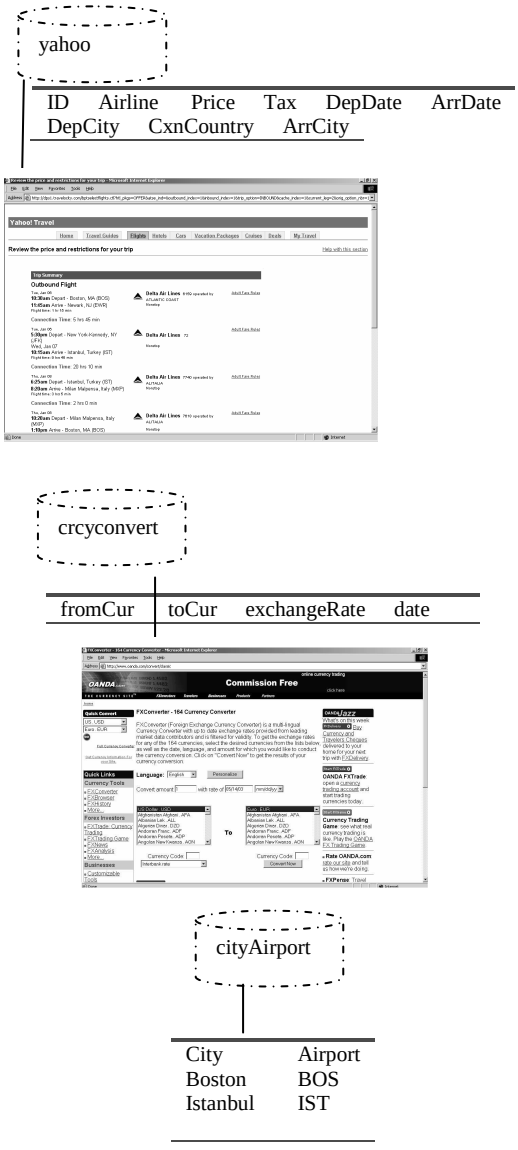
Ticket shipping cost is $20

Service fee of $5 is charged

Date is expressed in American style

Departure and Destination times are expressed as three letter airport codes

Currency is US dollar




## Context of User

Price means *round-trip final price* (including taxes, ticket shipment, and visa fees)

Date is expressed in European style

Departure and Destination times are expressed as city names

Currency is US Dollar

Direct air transit fee is applied if the plane has a connecting flight from Great Britain

Round trip price is twice the one-way price/tax/visa fees

Query

SELECT Price FROM yahoo

WHERE DepartureDate = “01/06/03” and ArrivalDate= “01/08/03” and DepartureCity= “Boston” and ArrivalCity= “Istanbul”;

## Context of VisaFees

Currency is British £

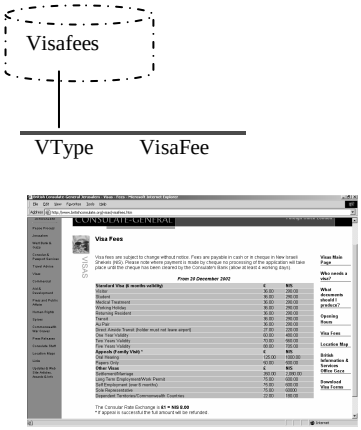



Figure 4.1 Airfare Example Scenario

.

This is an example of mediation in the case of contextual heterogeneities. The date and city name entities were *represented* differently in the source and user contexts, and the mediation engine detected and reconciled these conflicts. In the mediated query MQ1, we see examples of *dynamic* and *static* conversions. Date conflicts were resolved statically by converting the data values into the source context before the query is issued (e.g. 01/06/03 in Q1 became 06/01/03 in MQ1). City name conflicts, however, were resolved dynamically, with the help of cityAirport table, which is used to convert between city names and airport codes. Dynamic conversions are performed during query execution.

COIN prototype successfully deals with contextual heterogeneities as exampled here. It would not, however, be able to process ontological heterogeneities such as the conflicting *definitions* of price entity as shown in Figure 4.1 (price has conflicting definitions in yahoo and user contexts).

Despite knowing this limitation of COIN, let us assume that the user wants to learn the prices in addition to airlines and formulates the new query Q2 as follows:

```
Q2:   SELECT Airline, Price FROM Yahoo
      WHERE DepDate = "01/06/03" and ArrDate=  "01/08/03"
      and  DepCity= "Boston" and ArrCity= "Istanbul";
```

When this query is submitted to the COIN prototype, it returns the following result set[15]:

| **Airline** | **Price** |
|---|---|
| British Airways | 495 |
| Lufthansa | 525 |

This result set, however, is not semantically correct, because it fails to address the ontological conflicts concerning the term *price*. Yahoo reports prices as one-way and does not include extra costs such as taxes, shipment cost of the paper ticket, service fee, and any possible visa fee. The user, however, is expecting to see the final price of a round trip ticket including all kinds of costs.

In this case, our price sensitive friend would make a mistake by choosing British Airways over Lufthansa, because flying over Great Britain with British Airways would cost him a transit visa fee of £27, which would be more than the $30 difference.

If query Q2 were to be submitted to ECOIN system, it would first be rewritten into the following mediated query:

```
MQ2:  SELECT Airline, 2* (Price+Tax+ VisaFee*exchangeRate) + 25
      FROM yahoo, visafees, currencyconvert,
      (select Airport from cityAirport where city= "Boston") depCode,
      (select Airport from cityAirport where city= "Istanbul") arrCode,
      WHERE DepDate = "06/01/03" and ArrDate="08/01/03" and
```

[15] Since it does not have the capability to deal with *equational ontological conflicts* [see Chapter 6]

.

```
DepCity= depCode.Airport and ArrCity= arrCode.Airport
and CxnCountry= “Great Britain”;
UNION
SELECT Airline, 2* (Price+Tax) +25
FROM yahoo, visafees,
(select Airport from cityAirport where city= “Boston”) depCode,
(select Airport from cityAirport where city= “Istanbul”) arrCode,
WHERE DepDate = “06/01/03” and ArrDate=”08/01/03” and
DepCity= depCode.Airport and ArrCity= arrCode.Airport
and CxnCountry <> “Great Britain”;
```

and then would return the following answer, which is adjusted to reflect the user expectations:

| **Airline** | **Price** |
|---|---|
| British Airways | 1198 |
| Lufthansa | 1176 |

In the above mediated query MQ2, in addition to date and city name conflicts, the ontological price conflict is also resolved. The mediated query is a union of two queries because the price calculation depends on whether Great Britain, which imposes a transit visa fee for our user[16], is part of the flight or not. The first sub query in MQ2 corresponds to the case of having a connecting flight from Great Britain, thus adds the visa fee, adjusted in terms of currency, to the price along with tax, shipping and service fees. The price is then converted into a round trip price based on the contextual information provided by the user as shown in Figure 4.1. Note that the system aggregated the service fee and shipping cost by simplifying the arithmetic expression and simply added 25 to the price value.

As we will explain in more detail in the coming chapters, this capability of dealing with equational ontological conflicts is an important feature introduced by ECOIN through the use of symbolic equation solving techniques. Below we continue with another scenario that illustrates the issue of ontological heterogeneity in a corporate accounting setting.

## 4.1.2 Corporate Householding Scenario

In today’s rapidly evolving business environment, corporate group structures and the relationships between corporate entities are becoming more and more complex and difficult to understand. For legal purposes the corporate definition may include its branches, divisions, subsidiaries, and for tax purposes it may not. Interpretations of corporate structures depend on the context. In Figure 4.2, we illustrate such an example,

[16] See user context definition in Figure 4.1

in which the user interprets the financials of a company as the sum of financials of itself, its subsidiaries, branches and divisions. The data sources, however, report financials of a company by *excluding subsidiaries*: as the sum of financials of itself, branches and divisions. The user in Figure 4.2, poses the following naïve query:

Q3: SELECT Revenue FROM Financials

WHERE CorporateEntity = “IBM”;

**Context of FinancialsDB**

Financials of a corporate entity is the sum of financials of itself, branches, and divisions not including subsidiaries.

**r**

| CorporateEntity | Revenue |
|---|---|
| IBM | 77,966,000 |
| IBM Global Services | 36,360,000 |
| Lotus Development | 970,000 |
| IBM Far East Holdings | 550,000 |
| International Information Products | 1,200,000 |
| IBM International Treasury Services | 500,000 |
| General Motors | 177,828,100 |
| Hughes Electronics | 8,934,900 |
| Electronic Data Systems | 21,502,000 |
| … | … |

**RelationsDB**

| ChildEntity | ParentEntity | Relationship | % Ownership |
|---|---|---|---|
| Lotus Development | IBM | Subsidiary | 100 |
| IBM Far East Holdings B. V. | IBM | Subsidiary | 100 |
| International Information Products | IBM Far East Holdings B. V. | Subsidiary | 80 |
| IBM Global Services | IBM | Division | 100 |
| IBM Enterprise Investment | IBM | Division | 100 |
| IBM Software | IBM | Division | 100 |
| IBM Hardware | IBM | Division | 100 |
| IBM Global Financing | IBM | Division | 100 |
| IBM Germany | IBM | Branch | 100 |
| IBM France | IBM | Branch | 100 |
| IBM Finland | IBM | Branch | 100 |
| IBM Denmark | IBM | Branch | 100 |
| IBM Switzerland | IBM | Branch | 100 |
| IBM International Treasury Services | IBM Germany | Subsidiary | 33 |
| IBM International Treasury Services | IBM France | Subsidiary | 14 |
| IBM International Treasury Services | IBM Finland | Subsidiary | 10 |
| IBM International Treasury Services | IBM Denmark | Subsidiary | 18 |
| IBM International Treasury Services | IBM Switzerland | Subsidiary | 25 |

**Context of User**

Financials of a corporate entity is the sum of financials of itself, its subsidiaries, branches and divisions.

Query

SELECT Revenue FROM Financials

WHERE CorporateEntity = “IBM”;

Figure 4.2 Corporate Householding Example Scenario

.

The financials database would then return the following result set:

| **Revenue** |
|---|
| 77,966,000 |

This result above, however, is brought without paying any attention to the user context. If the same query were posed against the ECOIN mediation engine, the following mediated query would be executed:

MQ3:

SELECT $r_1$.Revenue + $r_2$.Revenue + 0.8 * $r_3$.Revenue + $r_4$.Revenue+ $r_5$.Revenue

FROM (select Revenue from r where CorporateEntity= “IBM”) $r_1$,

(select Revenue from r where CorporateEntity= “Lotus Development”) $r_2$,

(select Revenue from r where CorporateEntity= “International Information Products”) $r_3$,

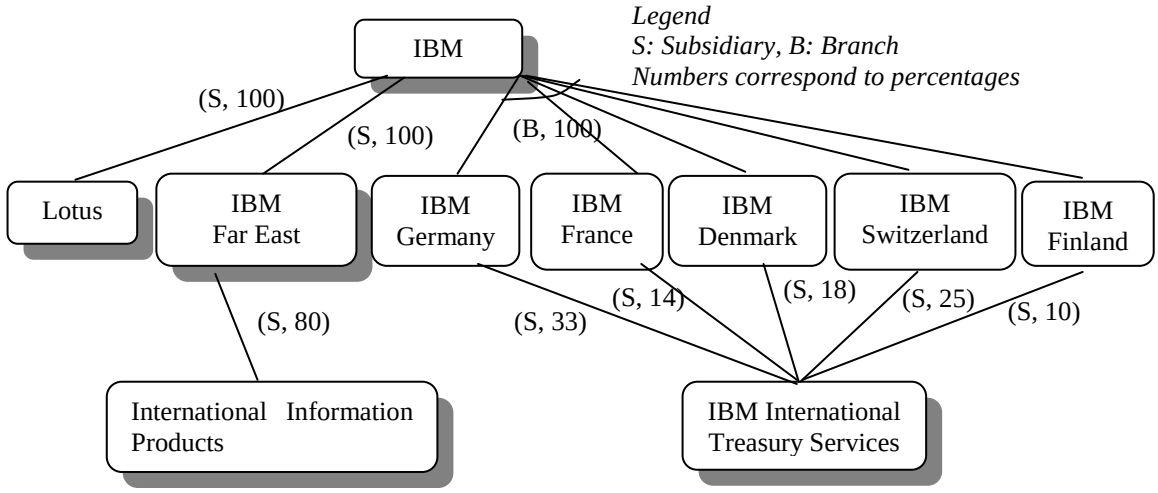
(select Revenue from r where CorporateEntity= “IBM Far East holdings”) $r_4$,

(select Revenue from r where CorporateEntity=“IBM International Treasury Services”) $r_5$

with the following result:

| **Revenue** |
|---|
| 116,756,000 |

The above-mediated query sums up the revenues of IBM, its subsidiaries, and the subsidiaries of its branches and divisions. This sum is constructed recursively by adjusting the financials of each corporate entity whether it is a corporate, branch, division or a subsidiary. The relationship and percentage of ownership information is obtained from Relationsdb, and the relevant part of the relationship tree is shown in Figure 4.3.



.

In these two examples, we explained how context mediation operates from the *user perspective*. The users (i.e. data providers and receivers) simply provide a declarative specification of *how data are interpreted* in sources and receivers, and a context mediator performs the reconciliations of conflicts. Next we explain how context mediation works from the *system perspective*.

## 4.2 Structural Elements of COIN Framework

In Chapter 3, we referred to Lenzerini's formalization of data integration systems as a triple (G, S, M) where

- G was the global schema, domain model, or ontology
- S was the source set, and
- M was the mapping between G and S [Lenzerini 03],

Using this notation, COIN framework can be roughly thought of as a quintuple (G, S, M, C, μ), where the additional two elements C and μ are defined as follows:

- C is the context multi-set, and
- μ is a mapping that assigns a context to each source.

In Figure 4.4, we show the interaction between the elements of COIN framework. In this figure the domain model G, context set C, and the source set S are shown in rectangles with rounded edges.

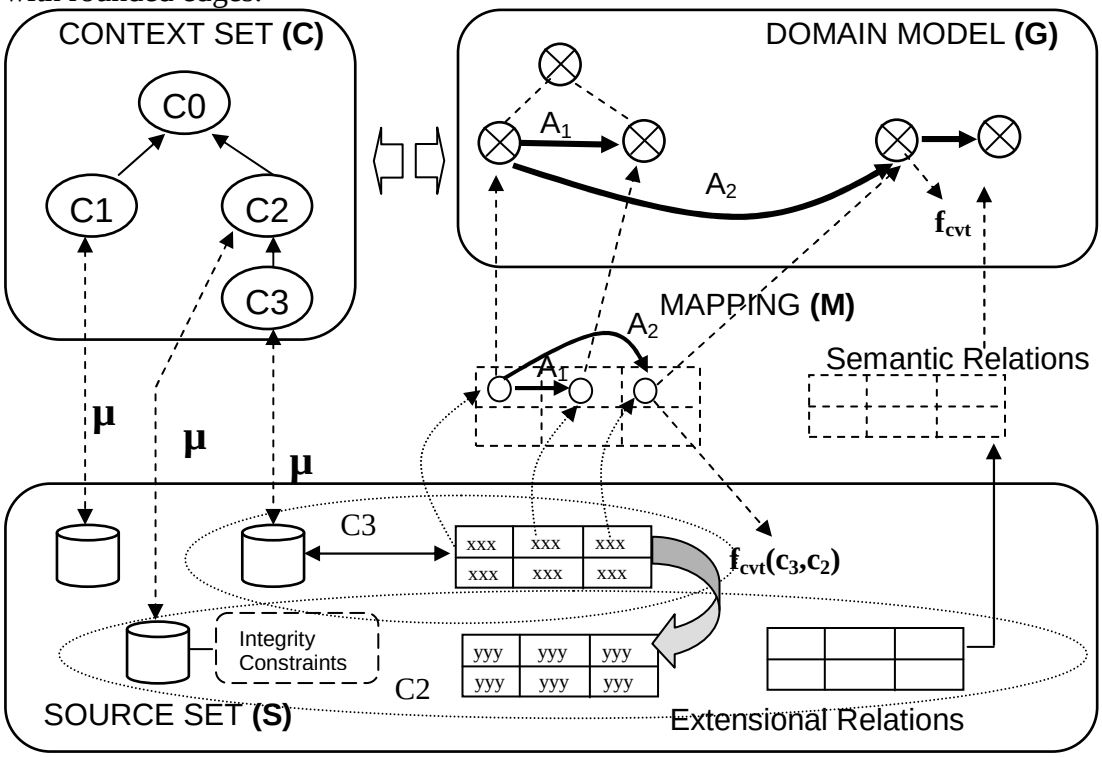



Figure 4.4 COIN Framework as quintuple (G, S, M, C, μ)

As shown in figure, mapping M (called *elevation axioms* in COIN*)*, maps extensional relations from the *source set* to semantic relations with the use of semantic types and attribute relations from the *domain model*. *Context multi-set* contains labeled sets of

context rules based on a framework defined in the *domain model*. Each labeled context set is assigned to a source according to their data semantics by function μ.Below we explain each element in more detail to clarify the structural elements of COIN.

### 4.2.1 Domain Model (G)

COIN has an object centric view of the world: it models information units as objects with unique and immutable object-ids. The domain model identifies object types (called *semantic types*) in a generalization hierarchy using the *is-a relationship;* some of the properties of objects using *attributes* and *modifiers*; and methods using *conversion functions*. While the domain model in COIN has a less elaborate collection of constructs than ontology languages such as Semantic Web's OWL classes [McGuinness and Van Harmelen 03], it nevertheless qualifies as an ontology language. In the rest of this Thesis, we will use domain model and ontology interchangeably.

As opposed to *primitive types* (e.g. strings, integers, and reals), semantic types in COIN are more abstract, and less implementation-oriented. Examples of semantic types in the form of rectangles with rounded edges can be seen in Figure 4.5, which depicts an ontology corresponding to the air fare example we provided in the beginning of this chapter. Instances of semantic types are called *semantic objects* (shown as circles within semantic relations in Figure 4.4), and their properties are represented by attributes and modifiers. *Is-a* relation defines the subtype-supertype relationship between semantic types.

Attributes define the state of an object or the relationship (similar to relationships in entity-relationship data modeling) between objects. For example, an object of a trip type would have a destination and origin city by definition, and this relationship is captured by the appropriate attributes as shown in Figure 4.5. Note that attributes are represented by solid arrows and their predictable names are omitted in Figure 4.5. Modifiers are special type of attributes used to capture sources of variations that affect the interpretation of a semantic object value[17]. They are shown with dashed arrows in Figure 4.4. For example, *moneyAmount* semantic type has a currency modifier, which is a source of variation for the values of its objects. Depending on the value of currency, *moneyAmount* objects may take different values. In COIN, modifiers are assumed to be independent of each other (i.e. orthogonal). This assumption simplifies the algorithm design permitting unordered application of conversion functions, which are defined per modifier. When there are dependent modifiers (i.e. conversion functions defined for those modifiers cannot be applied in any order), they have to be modeled as a single modifier.

Unlike primitive objects, semantic objects have values in a specified context. These values may be different depending on the modifier values, but they have to be semantically equivalent. This equivalency is established with the use of conversion functions that may also be viewed as *method definitions* for semantic objects. Conversion functions of a semantic object are defined per modifier, and are used to transform object values from one context to another. This is briefly illustrated in Figure 4.4, with the depiction of conversion function $f_{cvt.}$ In the figure, $f_{cvt}$ is shown in the domain model box to denote the method definitions, and also in the source set box to illustrate the conversion of values from one context to another. For example, a currency conversion

[17] Semantic object values are of primitive types, and correspond to values used in extensional relations.

function defined as a method of the semantic type moneyAmount and parameterized on its currency modifier values could be used to convert the values of moneyAmount objects from one context to another.

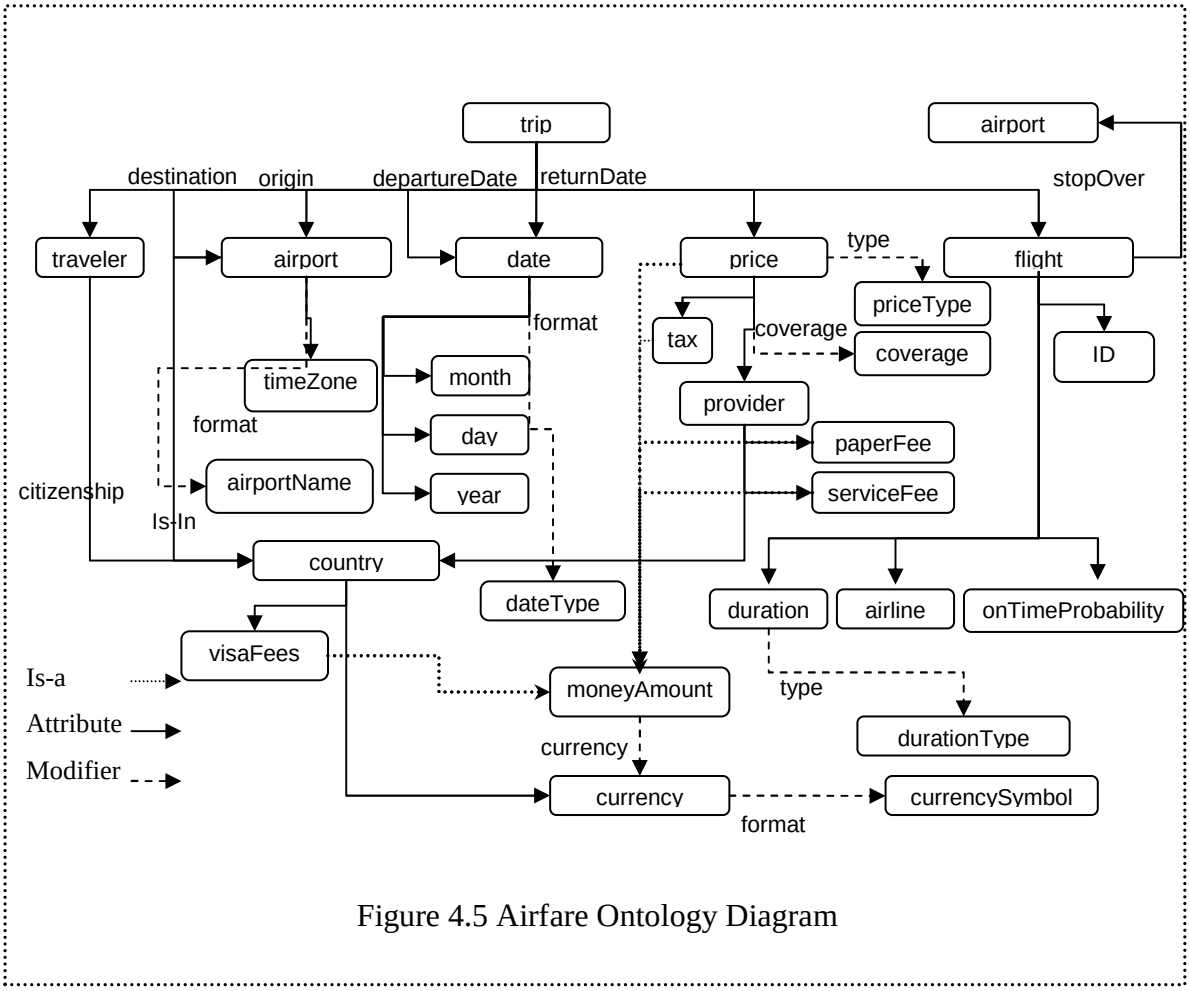



Figure 4.5 Airfare Ontology Diagram

Finally, we should mention that domain model language allows non-monotonic inheritance (*i.e. inheritance with overriding*) between semantic types. Conversion functions, attribute and modifier values can be polymorphically defined using this feature. For example, a conversion function defined for moneyAmount would apply to objects of its subtype (e.g. price). Similarly, modifier values defined for moneyAmount's currency modifier would be inherited by the objects of price unless they are explicitly overridden with its own definition of modifier values.

## 4.2.2 Source Set (S)

In COIN the canonical representation chosen for convenience is the relational data model. For that reason, the source set in COIN corresponds to a set of databases with their relations as shown in Figure 4.4 as extensional relations. Semi-structured data sources can also be used, but they have to be first converted into relational sources, for example, with the use of Caméléon wrapper engine. Domain and *integrity constraints* of

.

these relations are also represented within the source set as shown in Figure 4.4, and used for semantic optimization purposes. If for example, airfare provider yahoo did not sell tickets when the flight date was not in the current year, this integrity constraint could be used to return an empty answer to the following query when posed before year 2004:

Q4: SELECT Airline, Price FROM Yahoo

WHERE DepDate = "01/06/04" and ArrDate= "01/08/04"

and DepCity= "Boston" and ArrCity= "Istanbul";

### 4.2.3 Elevation Axioms (M)

Elevation axioms are used to relate sources and the domain model. Each *primitive relation* (i.e. ordinary database relations) in the source set is elevated to a *semantic relation*. This is accomplished by mapping each primitive object (i.e. data cells) in the primitive relation to a semantic object in the semantic relation. Skolem[18] functions are used to assign unique object ids to each semantic object. The skolem function has the following general structure:

$f_{\text{skolem}}(X) = skolem(Semantic\_Type(X), X, Context(X), Column\_Order(X), Primitive\_Relation(X))$

For example the primitive price object of type number from the primitive relation Yahoo shown in Figure 4.1, would be assigned the following unique object id:

$f_{skolem}(Price) = skolem(price, Price, c\_yahoo, 3, yahoo(I,A,Price,T,D,AD,DC,CC,AC))$

where I,A, Price,T,D,AD,DC,CC, and AC are logical variables correspond to the physical values of the yahoo relation.

Semantic relation Yahoo would then be constructed in COIN with the combination of each semantic object derived from a primitive relation as follows:

$yahoo'(f_{skolem}(ID), f_{skolem}(Airline), f_{skolem}(Price), f_{skolem}(Tax), f_{skolem}(DepDate), f_{skolem}(ArrDate), f_{skolem}(DepCity), f_{skolem}(CxnCountry), f_{skolem}(ArrCity))$

### 4.2.4 Context Set (C)

As we mentioned in the domain model section modifiers are special attributes that affect the interpretation of a semantic object value. The domain model in COIN defines what types of modifiers apply to which semantic types. Context set is a multi-set, which contains sets of rules that determine the values of modifiers in a particular context. For example, the modifier values for context of Yahoo (refer to Figure 4.1) based on the ontology from Figure 4.4 is shown in Table 4.1. The table shows the semantic types with their modifiers, and modifier values. Note that modifier values can be described via a

[18] Skolem functions are used to transform existentially qualified logical variables into universally qualified ones. Refer to [Goh 97] for more details.

.

variety of ways, but they must come from a shared domain that data sources and receivers agree.

| Semantic Type | Modifier | Value |
|---|---|---|
| Price | Type | Nominal |
| | Coverage | One Way |
| Date | Format | American |
| MoneyAmount | Currency | USD |
| Currency | Format | 3 character |
| Duration | type | hours |

Table 4.1 Extensional Context of Yahoo

The table corresponds to the *extensional context* of yahoo for it is a collection of physical values. *Intensional context* of yahoo is the labeled set of rules in the context multi-set that describe the context with logical expressions. For example the modifier values for price in the yahoo context (c_yahoo is used as the identifier) would be defined as:

```
modifier(price, Object, type, c_yahoo, Modifier) ←
        cste(priceType, Modifier, c_yahoo, "nominal").

modifier(price, Object, coverage, c_yahoo, Modifier) ←
        cste(coverage, Modifier, c_yahoo, "one-way").
```

These definitions above correspond to *static* modifier declarations whose values are constant (i.e. “nominal” and “one-way”). It is also possible to define modifiers whose values are variables. Below is an example of a *dynamic* modifier declaration for the currency modifier of moneyAmount semantic type:

```
modifier(moneyAmount, Object, currency, c_intl, Modifier) ←
        attr(Object, provider, Provider),
        attr(Provider, locatedin, Country),
        attr(Country, officialCurrency, Modifier).
```

In this modifier declaration the value of currency modifier is obtained by finding the provider from the price object, the country from the provider object, and finally finding the official currency from the country object. Note that modifiers themselves are semantic objects, therefore may have their own modifiers. An example of this is shown in Figure 4.5 and Table 4.1, in which the currency type has a modifier called format.

## 4.2.5 Context Assignments (μ)

Contexts labeled in the context set are assigned to relations with the mapping μ. In the elevation axioms section we have shown how the sources and the ontology are linked

.

with the construction of semantic objects. Context assignments are done during the semantic object construction as seen in the example we repeat below:

$f_{skolem}(X) = skolem(Semantic\_Type(X), X^{19}, \mathbf{Context(X)}, Column\_Order(X), Primitive\_Relation(X))$

For example, c_yahoo context identifier is assigned to the price data cell from the yahoo primitive relation as follows:

$f_{skolem}(Price) = skolem(price, Price, c\_yahoo, 3, yahoo(I,A,Price,T,D,AD,DC,CC,AC))$

With this mapping structure data cells within a single relation can be assigned different context identifiers.

# 4.3 Query Answering in the COIN Framework

Queries in COIN are formulated in SQL in a receiver context[20,21], and refer to individual database schemas and/or views. SQL queries input by users are translated into clausal form (i.e. Datalog, which is the *logical* equivalent of SQL) in COIN. For example query Q2 would be translated into CQ2 shown below in Datalog like notation[22]:

(Naïve query) NQ2: ← answer(Airline,Price)

answer(Airline,Price) ←
    yahoo(_,Airline, Price, _, "01/06/03", "01/08/03" , "Boston", _, "Istanbul").
Context: c_user

This query is named as a *naïve query*, because the direct execution of it would ignore potential semantic conflicts and would most likely return inaccurate answers. Therefore, this query is converted into a *well-formed*[23] query, through steps explained in [Goh 97]. This well-formed query refers to semantic relations instead of primitive relations, and therefore uses the *value(X, C, Y)* notation to refer to the primitive value Y of semantic object X in context C. The example query, NQ2, can be written as a well formed query as follows:

(Well formed query) WQ2: ← answer(VAirline,VPrice)
answer(VAirline,VPrice) ←
    yahoo′(_[24],Airline′, Price′, _, DDate′, ADate′ , DCity′, _, ACity′),
    value(Price′, c_user, VPrice)[25],
    value(Airline′,c_user, VAirline),
    value(DDate′,c_user, "01/06/03"),

[19] This value refers to primitive relation
[20] Can also be called user or query context
[21] Like data sources, users have their own contexts defined by modifier value assignments
[22] For easier readability we will use Datalog style instead of the style adopted in [Goh 97].
[23] This name is adopted taken from [Goh 97].
[24] _ in logic programming corresponds to anonymous variables whose names are not needed.
[25] Read as "the value of semantic object Price′ in context c_user is Price".

.

value(ADate′,c_user, “01/08/03”)
value(DCity′, c_user, “Boston”)
value(ACity′, c_user, “Istanbul”).

The above well formed query incorporates the receiver context, and rewrites the naïve query using the semantic relations corresponding to primitive relations(yahoo′ vs. yahoo). Because the objects of semantic relations are semantic objects, they are mapped to their primitive values in the user context (c_user) through the use of value predicate.

This well formed query then needs to be rewritten into a mediated query to obtain the *intensional answer* to the original query. *Abductive reasoning* is chosen here for it provides the desired intensional answer, as opposed to the extensional answer that would be obtained from deductive reasoning. Next, we explain the details of abductive reasoning from operational point of view.

### 4.3.1 Overview of Abduction in COIN

Like induction, and deduction, abduction is an inference technique used in logic. It is a form of hypothetical reasoning that offers explanatory facts from observed ones and a set of rules [Denecker and Kakas 02]. In the simplest case, given the rule “Y implies X”, and the observation X, abduction infers Y as a possible explanation of X.

An *abductive programming framework*, consists of a set of rules known as the theory (e.g. {“Y implies X”}), possible explanations known as abducibles (e.g. Y) and integrity constraints (e.g. Y must be integer) Given a query Q, the abductive reasoning task is to find a set of explanations from the set of abducibles such that they are consistent with the theory and integrity constraints.

In order to use abductive reasoning, COIN framework (G, S, M, C, μ) is first converted into an abductive framework by constructing the

- theory from the union of domain model G, mappings M, context set C, and context mappings μ, which constitute a set of rules;
- integrity constraints from the integrity constraints defined in the source set S; and
- abducibles from extensional predicates (e.g. yahoo relation) and built in predicates that is admitted by the query language (e.g. +, >, etc.)

Then given a well formed clausal query Q, abductive reasoning algorithm progresses to find a set of extensional and built in predicates as an answer. The details of how abductive reasoning works can be found in [Denecker and Kakas 02]. The end result of abduction in COIN is a mediated query expressed only using the extensional relations, and built-in features that are admitted by SQL. It is also considered as the intensional answer to the query issued by the user, for it expresses the answer in terms of predicates not with a set of facts. This intensional answer can then be optimized, and executed against the sources to retrieve extensional answers.

## 4.4 ECOIN compared to COIN

As we explained in this chapter COIN is a realization of the Context Interchange strategy first articulated in [Siegel and Madnick 91, Sciore et al. 94] in the form of a data model, reasoning algorithm, and a prototype implementation. ECOIN is another

realization of the same strategy, with differences in representation, reasoning and the prototype. Below we analyze these differences one by one.

### 4.4.1 Representational Differences

ECOIN improves COIN by providing a solution for representing and reasoning with equational ontological heterogeneities. We introduce a conceptual transformation of ontological heterogeneities into contextual heterogeneities by using loosely defined terminology in the shared ontology. Yet, the way ontological conflicts are represented in ECOIN is not unlike the way contextual heterogeneities are represented in COIN. Representational differences between the two can be found in the way conversion functions are represented in ECOIN. Unlike COIN, conversion functions in ECOIN constitute a bidirectional graph. Combined with the new reasoning capabilities, this new representation considerably cuts down the need to define new conversion functions.

In ECOIN, we address the merging of disparate applications which is non-existent in COIN. We define a completely new and backward compatible representation of merging axioms, which allows scalable construction of ECOIN applications.

### 4.4.2 Reasoning Differences

The major novelties introduced by ECOIN are in the reasoning algorithm and prototype implementation. ECOIN introduces a symbolic equation solver encoded using a declarative custom constraint definition language called Constraint Handling Rules (CHR). We combine abductive and constraint logic programming to intertwine query mediation with symbolic equation solving, which is essential in reasoning with equational ontological conflicts. ECOIN also provides the functionality to operate on the conversion graph with the integration of Dijkstra's scalable shortest path algorithm into the reasoning process.

Furthermore, the reasoning algorithm is expanded to deal with the new demands of merging disparate ECOIN applications. This expanded reasoning algorithm is a first attempt in dealing with virtual applications that do not physically contain ECOIN axioms, but links other applications using the ECOIN merging framework.

### 4.4.3 Prototype Differences

ECOIN offers numerous improvements over the prototype implementation of COIN. First, it actually implements many of the missing features of COIN prototype such as inheritance between semantic types, context identifiers, and modifiers. Second, ECOIN prototype implementation is more modular compared to COIN, as it has a cleaner separation between different system components. This makes it easier to understand and build on top of the ECOIN prototype for future students working on the project. The query optimizer and execution engine of ECOIN is built on a new capability based optimizer, and has been more reliable compared to COIN optimizer and execution engine. Furthermore, ECOIN offers graphical metadata management, application building and merging tools for end users.

In the coming chapters we provide a detailed account of the features of ECOIN.

# Chapter 5

# Extended COntext INterchange

Extended Context Interchange (ECOIN) provides a knowledge representation framework and a reasoning platform for integrating heterogeneous information systems. In this chapter, we describe the ECOIN knowledge representation framework in detail. Before doing that we find it useful to provide background on some of the key concepts behind the ECOIN representation framework, namely, *context*, *ontology* and *logic programming.*

## 5.1 Context

With the globalization of information over the internet, recognizing the role of context in achieving semantic interoperability among heterogeneous and autonomous systems has become an important endeavor. In this section we provide an overview of studies on context that can be found in the literature of philosophy, and artificial intelligence (AI), including knowledge representation, natural language processing, and intelligent information retrieval.

According to Collins Cobuild English Language Dictionary the context of something consists of "the ideas, situations, events, or information that relate to it and make it possible to understand it fully" [Akman 02]. In [Sperber & Wilson 86], context is defined as "the set of premises used in interpreting an utterance, a psychological construct, a subset of the hearer's assumptions about the world". In [McCarthy 93], no definition of context is offered, for they are viewed as rich objects that cannot be completely described [Guha 91].

In the field of philosophy, study of context goes back to 1898 to philosopher Charles Peirce, who describes making meta level assertions about nested propositions[26] as well as the rules of inference for importing and exporting information into and out of the contexts [Sowa 97]. Another philosopher Mario Bunge provides a formal definition of context in [Bunge 74] as follows:

[26] This is known as *reification* in the AI community.

.

"DEFINITION 2.10 The ordered triple C=<S,P,D> is called a (conceptual) context (or frame) iff S is a set of statements in which only the predicate family ℙ occur and the reference class of every P in ℙ is included in the universe or domain D ⊆ Ω."
This definition defines context as a *set of statements* constructed by using a set of *designated predicates* and a *domain* those predicates draw upon. Consider for instance the conceptualization of the price of an airfare with the predicate set P={price, currency, scalefactor}, and domain D ={$D_{airfareID}$, $D_{price}$, $D_{currency}$, $D_{scalefactor}$} which corresponds to domains used in the predicate declarations (e.g. $D_{currency}$ = {USD, EUR, …, JPY}. The context as defined by Bunge in this case would correspond to a set of equations that can only be constructed with a designated set of predicates and their domains. One example would be:
C = <**{**currency(LH421, USD), scalefactor(LH421, 1),…**}**, **{**price, currency, scalefactor**}**, **{**LH421,LH422, NW030,…, 0,0.01,0.02,…, USD, EUR,…ITL,1,1000**}**>
where S describes the currency and scalefactor of each airfare, P is the set of predicates that can be used in S, and D is the domain of the predicate declarations. This definition constitutes the basis of context representation in [Lee 96].

In AI, the main motivation for studying formal contexts is to resolve the 'problem of generality' [McCarthy 87]. Although computers can beat the best human player in chess, they lack the ability to generalize specialized knowledge. Thus, the study of context in AI concentrates on finding a unified formal framework for representing and reasoning with context. This may be quite challenging (and even arguably unproductive [Hirst 00]), given that context is used for different purposes in different sub fields of AI. In knowledge representation and reasoning, for example, context is thought of as an agent's partial representation of the world, whereas in natural language processing, context is conceived of as a collection of features of the location in which an agent produces a linguistic expression, and is therefore assumed to be related to the state of the world [Bouquet et. al. 01]. Along the same lines, Akman, drawing from literary theory and social sciences, views context as a social construct and asserts that interpretation is possible only within shared contexts [Akman and Surav 97].

The formalization efforts of context are categorized into two groups by [Bouquet and Serafini 03]: *the Propositional Logic of Context* (PLC) [Buvač and Mason 93] and *Local Models Semantics* (LMS) [Ghidini and Giunchiglia 01]. The comparison of these two approaches is analogous to that of tight vs. loose coupling approaches in database integration and we find it useful to explain each in more detail.

## 5.1.1 Tightly Coupled Contexts

PLC is an example of this approach, which was also called "divide and conquer" in [Bouquet et. al. 01]. We call this approach tightly coupled because of the *existence of a unique global vocabulary* that ties all the contexts together. McCarthy's efforts fall under this category, and so does the approach used in CYC [Lenat et al., 1990; Guha, 1991], as a way of partitioning a global theory of the world with two levels of nesting: micro-theories and the default outer level.

.

In this approach axioms and statements are true only in a context. This is expressed by a *modality*[27] called ist(c,p)[28]. For example,

$c_0$: ist(context-of("Sherlock Holmes stories"), "Holmes is a detective").

means that the statement "Holmes is a detective" is true in the context of Sherlock Holmes stories. The preceding $c_0$ denotes that this statement is asserted in an outer context, thus points out to the nested composition of context dependent statements. Formulas between contexts can be related together with the use of lifting axioms.

In [Buvac 98], one of the best representatives of this approach, an example of reasoning with Navy and General Electric (GE) databases is given. In this example, databases differ on the definition of engine prices, which include assortment of spare parts and warranty in the Navy database, in addition to GE's reported plain engine price. Contexts defined in this example are $c_{GE}$, $c_{navy}$ corresponding to the GE and navy databases and $c_{ps}$, the problem solving context. The details of this example, (i.e. the query posed in the problem solving context, the existing facts expressed in their own context, and lifting axioms that define translations between different contexts) are shown in Table 5.1.

| |
|---|
| **Query** |
| $c_{ps}$: ist($c_{navy}$, price(FX-22-engine, \$3611K)) |
| **Facts** |
| ist($c_{GE}$, price(FX-22-engine, \$3600K)). |
| ist($c_{GE}$, price(FX-22-engine-fan-blades, \$5K)). |
| ist($c_{GE}$, price(FX-22-engine-two-year-warranty, \$6K)). |
| ist($c_{navy}$, spares(FX-22-engine,FX-22-engine-fan-blades)). |
| ist($c_{navy}$, warranty(FX-22-engine,FX-22-engine-two-year-warranty)). |
| **Lifting axioms** |
| value[29]($c_{GE}$, price(x)) = GE-price(x) |
| value($c_{navy}$, price(x)) = GE-price(x) + GE-price(spares($C_{Navy,}$ x)) + GE-price(warranty($C_{Navy}$, x)). |

Table 5.1 Navy and General Electric Integration Example

## 5.1.2 Loosely Coupled Contexts

LMS is an example of this approach, which was also called *"compose and conquer"* in [Bouquet et. al. 01]. We call this approach loosely coupled because there is *no such a thing as a global theory* of the world, but only many local theories. Unlike the tightly coupled approach, contexts in loosely coupled approach are autonomous theories with no predefined common vocabulary. Relations between contexts, are established on a peer-to peer basis, as a collection of constraints on what can (or cannot) be true in a context given that there is some relation with what holds in another context. This is depicted in Figure 5.1, with each rectangle corresponding to local theories with different contexts and dotted arrows establish the links between their contexts.

[27] The classification of propositions on the basis of whether they assert or deny the possibility, impossibility, contingency, or necessity of their content.
[28] Read as "p is true in context c"
[29] value(c,t) is a function which returns the value of term t in context c

.

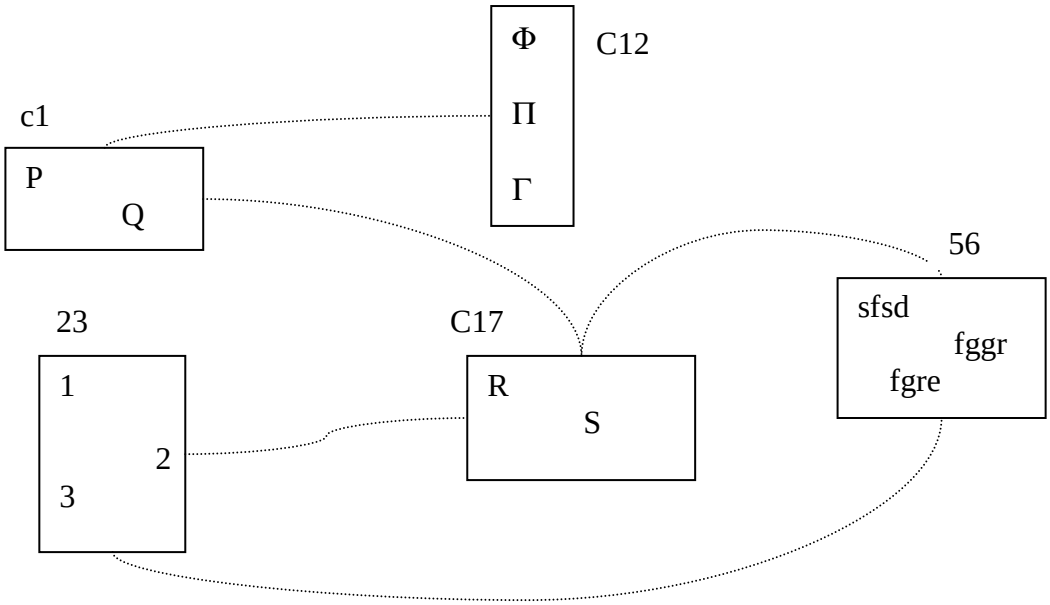



Figure 5.1 Loosely coupled contexts (adopted from [Bouquet et. al. 01])

This approach has been used in [Ghidini and Serafini 98, 00], in integrating information systems. They provide an example that integrates the databases of four fruit sellers with different contexts. Conflict resolution between contexts is done pair wise for each database, since they do not subscribe to a common global theory. In the example, one of the sellers (1) provides fruit prices without including taxes, the other denoted as the mediator (m) considers prices with taxes (7% percent). This conflict is resolved by defining a view constraint as following:

1: has-price(x,y) $\rightarrow$ m:$\exists$y′ has-price(x,y′) $\wedge$ y′ =y +(0.07*y)

This view constraint establishes the link between differing price definitions of source 1 and mediator m.

While loosely coupled approach to modeling contexts offers more flexibility in dealing with contextual disparities among data sources, it suffers from the scalability issues since constraints that relate context of each source should be defined on a peer to peer basis. This requires each source to know about all other sources, which may be quite costly. Tightly coupled approach to modeling contexts, on the other hand, enables one to relate contexts of sources (i.e. by using lifting axioms) more generally with the power of a shared language, but it suffers from the flexibility issues as the language evolution has to be coordinated between sources.

# 5.2 Ontology

In recent years, the study of ontology, with its roots in philosophy, has become intertwined with the development of artificial intelligence and of information systems science [Smith and Belty 01]. Although the term ontology assumes a different meaning in philosophy (i.e. the metaphysical study of the nature of being and existence), in computer science it corresponds roughly to an “agreement about a shared, formal, explicit and

partial account of a conceptualization" [Gruber 93, Guarino and Guiretta 95, Ushold and Gruninger 96, Spyns et. al 02]. Ontologies can range from simple glossaries or relational database schemata to a hierarchy of concepts related by subsumption relationships to formal logical theories that specify relations and constraints between terms as well as inference rules.

In information integration, ontologies are increasingly being used as a unifying framework for translating between different models of heterogeneous data sources to achieve semantic interoperability. Information integration projects such as SIMS [Ariens et. al 96], CARNOT [Collet et. al 91], InfoSleuth [Woelk and Tomlinson 94], OBSERVER [Mena et. al 96], Information Manifold[Levy et. al 95], COIN[Goh 97] are all ontology-based approaches [Wache et. al 01]. A detailed study of ontologies in information systems can be found in [Smith 03] and [Guarino 98].

Among various studies found in the literature we find Guarino's in depth study insightful and most relevant to this Thesis. In [Guarino 98], an ontology is defined as follows and illustrated in Figure 5.2:

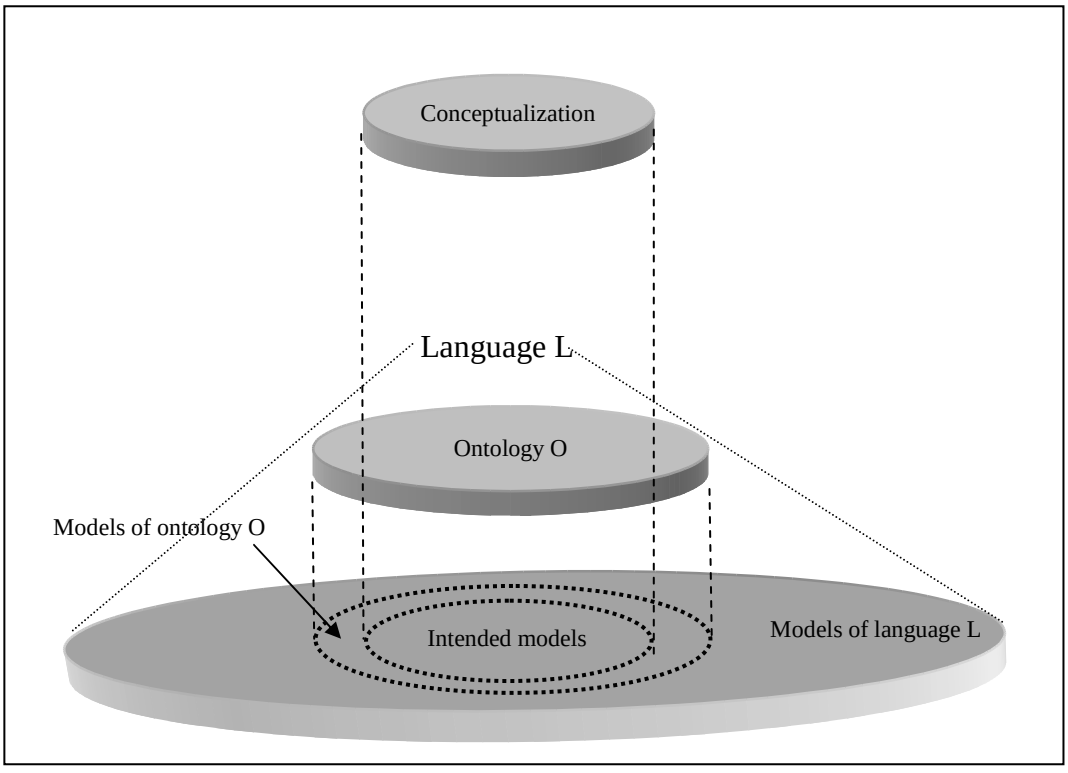



Figure 5.2. Ontology as a coarse specification of a conceptualization

"An ontology is a logical theory accounting for the intended meaning of a formal vocabulary, i.e. its ontological commitment to a particular conceptualization[30] of the world. The intended models[31] of a logical language using such a vocabulary are

[30] a set of informal rules that constrain the structure of a piece of reality

[31] A model is an interpretation (i.e., an assignment of truth values to symbols) of a set of sentences such that each sentence is "true".

constrained by its ontological commitment. An ontology indirectly reflects this commitment (and the underlying conceptualization) by approximating these intended models."

The formal details of this description are beyond the scope of this Thesis, and can be found in [Guarino 98]. Intuitively, however, this definition underlines an important property of ontologies that they are imprecise specifications of conceptualizations; therefore they also admit unintended models of a language depending on how fine-grained they are specified. In Figure 5.2, an ontology is shown to restrict the models of a language, yet this restriction is not precise enough to exactly correspond to the intended models of the conceptualization it specifies. In Figure 5.3, we give an example involving multiple conceptualizations and one ontology to approximate the intended models of both conceptualizations.

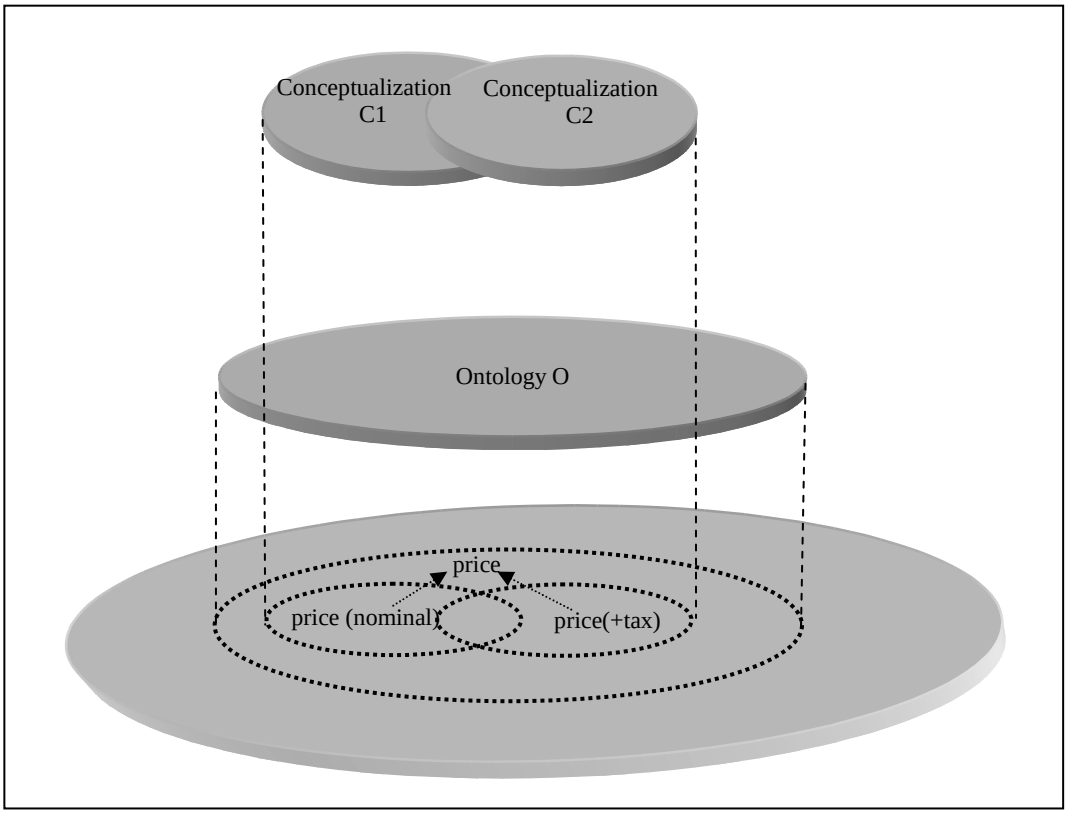



Figure 5.3. Ontology as a coarse specification of a conceptualization

The ontology in this example has to be less restrictive than either of the conceptualizations, since it has to be able to specify both. For example, even though the two conceptualizations may have different definitions for price (nominal vs. including tax) in their intended models, the ontology created to approximate these conceptualizations may instead adopt a more general definition of price that subsumes both variations as special cases. We further advance this idea, and suggest that the variations can be represented using contexts, as will be explained in more detail in coming chapters.

.

## 5.3 Logic Programming

Logic programming, developed in the early 1970s, is a declarative method of knowledge representation and programming based on first-order logic [Kowalski 1974]. A general logic program can be viewed as a collection of rules (or clauses) of the form [Baral and Gelfond 94]:

$$A_0 \leftarrow A_1, \ldots, A_m, \text{not } A_{m+1}, \ldots, \text{not } A_n$$

Each $A_i$ is a literal (or atom) in the form $p_i(a_{i1}, \ldots, a_{in})$, where $p_i$ is a predicate symbol and $a_{ij}$ are terms. *Not* is a logical connective called negation as failure in [Clark 78], $A_0$ is the head (or conclusion), the right hand side of the rule is the body (or premise). In the special case of body being empty, the head is also called a *fact*.

Rules or literals that do not contain any variables are called *ground*. The set of all ground literals in the language of a logic program is called its *Herbrand base*. The set of all possible terms that the theory can make assertions about are called its *Herbrand universe*. An *interpretation* is an assignment of literals to truth values. A *model* of a logic program is an interpretation, which satisfies all of its rules.

General logic programs that do not have negative literals are called definite programs, also known as, Datalog. Database relations can be defined in two ways in Datalog: *extensional database* (EDB) relation is the set of ground facts often stored in a database, whereas *intensional database* (IDB) relations are defined by logical rules. In the relational model all relations are EDBs and view definitions are analogous to IDBs.

The semantics of a logic program depends on how they define the satisfiability of the rules. The meaning of a Datalog program is defined in three ways in [Ullman 91]. *Proof theoretic* interpretation is the set of facts that can be derived from the rules in a logic program (i.e. forward chaining). The *model theoretic* interpretation corresponds to finding an assignment of truth values to all variables that makes all rules true. The *computational* interpretation is about designing an algorithm for executing rules to determine whether a predicate is true or false. For definite logic programs all of these interpretations coincide with one another.

The semantics of a general logic program, however, is complicated by the interpretation of negation [Apt and Bol 94]. Namely, how does one evaluate “not Q”? One of the first attempts for the interpretation of negation is *Clark’s Completion Semantics*, which, informally stating, replaces the implications by equivalences [Clark 78, Lloyd 84]. In *negation as finite failure*, not Q is a consequence of a program if Q finitely fails [Clark 78]. Finally, using the *closed world assumption*, not Q is a consequence of a program if Q cannot be proven. The relationships between these interpretations are examined in detail in [Apt 90].

Due to some inadequacies of the above stated approaches to negation, other approaches have been suggested. *Perfect model semantics*, for example, is defined for stratified programs (i.e. programs which can be decomposed into different layers), where predicates defined in a given layer cannot depend negatively on predicates defined in lower layers [Castro and Warren 00]. *Stable model semantics* and *well founded semantics* try to extend the perfect model semantics to general logic programs, the first one taking a possible worlds approach by assigning a set of models to a program, the latter by assigning a unique three-valued interpretation to each program.

.

In logic programming languages such as Prolog, definite logic programs extended with negation, built-in predicates (i.e. =, <, and so on), and function symbols are allowed. Inclusion of built-in predicates and functions require that a logic program is safe (i.e. derives a finite set of answers), which is easily verified using a safety criteria discussed in [Ullman 91]. Queries can be evaluated bottom up or top-down, by inferring new facts from the existing facts or recursively transforming a goal into a series of subgoals that terminate in a fact. Detailed analysis of query evaluation techniques can be found in [Dantsin et. al. 01].

ECOIN data model is based on Datalog with negation extended with built-in predicates and function symbols. We explain the details of ECOIN next.

# 5.4 ECOIN Knowledge Representation

In this section we formally introduce knowledge representation in ECOIN, which is built on top of COIN. For completeness, definitions which are inherited unchanged from COIN will be stated here as well.

ECOIN uses first order logic (FOL) as the language of representation, departing from COIN's use of F-Logic and Gulog inspired language (COINL) [Goh 97]. We choose FOL for a number of reasons. First, (FOL) has been around longer than both F-Logic and Gulog, and is better understood theoretically. Second, FOL can be used both for knowledge representation and programming, allowing us to easily make the transition from theory to implementation. Furthermore, this choice was influenced by the experience and familiarity of our research group members with FOL compared to the COINL.

As mentioned in the previous section, it is difficult to come up with an all-encompassing definition for context and ontology, for they may have different definitions depending on the purpose. The definitions in this section are given for the purpose of formalizing ECOIN, and do not intend to provide generalized definitions.

Our structure of introducing knowledge representation in ECOIN will be by providing the syntax and semantics of its language, which will culminate in the construction of an ECOIN framework. Definitions will be followed by informal restatements when need be, and also by examples to clarify the meaning. This framework introduced in this chapter is further extended in the Chapter 7, with merging related constructs.

## 5.4.1 Basic Concepts

In this section we provide the definitions of basic constructs used in ECOIN.

**Definition** (Constructs)

- A *source* is a set of predicates intensionally describing database relations.
- A *context identifier* is a unique constant.
- A *primitive type* is a data type with a materialized domain in native sources. Instances of primitive types are called *primitive objects*. The *value of a primitive object* is equivalent to itself (i.e. object identifier and value are the same).
- A *semantic type* is a conceptual data type without a materialized domain. A *semantic object* is an instance of a semantic type. The *value* of a semantic object is obtained by a function $f$:$C \times S_O \rightarrow P_O$, where $S_O$ is the domain of semantic

.

objects (identifiers), C is a set of *context identifiers* and $P_O$ is a set of primitive objects.

- An *attribute* specifies a property of a semantic type, and can be viewed as a function $f$: S→S, where S is the set of semantic types.
- A *modifier* m is a *designated* attribute. The decision to designate a certain attribute as a modifier depends on whether that attribute is implicitly specified in some sources as contextual knowledge and if the value of that attribute functionally determines the value of a

**Examples:**

- An example source would be the yahoo database shown in the airfare example, with yahoo, cityAirport, crcyconvert relations.
- An example context identifier would be c_yahoo as in the airfare example.
- An example of a primitive type in SQL would be varchar, real, and integer. Their values such as "Smith", 2.1, 1 would be examples of primitive objects.
- An example of a semantic type would be the concept of number independent of its representation. Similarly, number two without any specific representation would be an example of a semantic object. The object identifier of a semantic object will be described later. The value of the semantic object "number two" in context c_yahoo would be 2.
- Example attributes would be the *price* property of semantic type product, and *currency* attribute of semantic type price. (Note that price is used both as a property and semantic type identifier in this example, which is legitimate in ECOIN)
- An example modifier would be the *currency* attribute of price semantic type. The currency modifier of price could take different values (e.g. USD, EUR) in different contexts, which would affect the value of semantic price objects.

## 5.4.2 Declarations

This section contains basic declaration syntax used in the ECOIN framework. In definitions below $\tau,\tau'$ are semantic types; t, t′ are semantic objects; a is an attribute symbol; m is a modifier name; c is a context identifier; $t_p$, $t'_p$, mvs, $mv_t$ are primitive objects; $L_1$, …, $L_n$ are atoms or user defined literals.

**Definition** (Declarations) A declaration is defined as follows

- A *sub-type relationship* ($\tau$ is a sub-type of $\tau'$):

  $$is\_a(\tau,\tau')$$

  In this case, $\tau,\tau'$ can also be context identifiers.
- *Attributes* of a semantic type $\tau$:

  $$attributes(\tau,[a_1,\ldots a_n])$$
- *Modifiers* of a semantic type $\tau$:

  $$modifiers(\tau,[m_1,\ldots,m_k]):$$
- A *source relation*:

  $$relation(D,R,A,S)$$

where D is the database identifier, R is the relation name, A specifies if the relation is abducible, S is the schema of R as a set of pairs in the form of [primitive object, primitive type].

- An *attribute atom*:

$$\text{attribute}(t,a,t') \leftarrow L_1, \ldots, L_n.$$

- A *modifier atom*:

$$\text{modifier}(\tau,t,m,c,t') \leftarrow L_1, \ldots, L_n.$$

- A *value function* $f$:$S_O \times C \rightarrow P_O$ where $S_O$ is a set of semantic objects, $P_O$ is a set of primitive objects and C is a set of context identifiers:

$$\text{value}(t, c, t_p)$$

**Examples:**

- is_a(product, basic): basic is a supertype of product (or product is a subtype of basic).
- is_a(price, monetary_value): monetary_value is a supertype of price.
- is_a(c_yahoo, c_usa): c_yahoo context is a subtype of c_usa context
- attributes(product, [country, price]): semantic type product has the attributes country and price.
- modifiers(price, [currency, type]): semantic type price has the modifiers currency and type(specifies whether the price is nominal or includes tax etc.).
- Relation: relation(yahooDB, cityAirport, i, [[city,string],[airport, string]]).
  where yahooDB is the database name, cityAirport is a relation in this database, i denotes that the relation is abducible, (e would denote that the relation is not abducible -- e.g. a view--), city and airport of type string are the column names of the cityAirport relation.
- Simple attribute declaration:

  $\text{attribute}(\text{Product}, \text{price}, \text{Price}) \leftarrow r'_1(\text{Product}, \text{Price}).$

  where $r'_1$ is a semantic relation, price is the attribute name, Product and Price are semantic objects
  This declaration defines how the price attribute of semantic object *Product* can be obtained from the semantic relation $r'_1$. We call it simple since it involves only one semantic relation.
- Complex attribute declarations with semantic joins:

  $\text{attribute}(\text{Product}, \text{country}, \text{Country}) \leftarrow r'_1(\text{Product}_1, \text{Price}),$
  $r'_2(\text{Product}_2, \text{Country}), \text{value}(\text{Product}, c, \text{Product}_p),$
  $\text{value}(\text{Product}_1, c, \text{Product}_p), \text{value}(\text{Product}_2, c, \text{Product}_p).$

  where $r'_1, r'_2$ are semantic relations, $\text{Product}_1$, $\text{Product}_2$, Country are semantic objects, $\text{Product}_p$ is a primitive object, c is a context identifier.
  This declaration means that the country attribute of a product can be obtained by joining two semantic relations $r'_1$ and $r'_2$ on the $\text{Product}_1$ and $\text{Product}_2$ semantic objects. Value functions are used to enforce that the join objects (i.e. $\text{Product}_1$ and $\text{Product}_2$) and the object under consideration (i.e. Product) have the same primitive values in the same context.

- Static Modifier:

  $\text{modifier}(\text{price}, \text{Price}, \text{currency}, \text{c\_usa}, M) \leftarrow$

cste(currencyType, M, c_usa, "USD").

where cste is a utility function that builds a semantic object from static values. The required inputs for this function are the type of the modifier object (currencyType), the modifier identifier (M), context in which the modifier value is being declared (c_usa), and the static value of the modifier in that context ("USD").

- Dynamic Modifier:

modifier(price, Price, currency, c_world, M) ←
attribute(Price, product, Product),
attribute(Product, country, Country),
attribute(Country, officialCurrency, M).

In this example, the modifier represented by the semantic object M, is being assigned a semantic object directly (as opposed to building one through the use of cste function) by using the attribute functions. Given the semantic object Price, first its product[32] attribute is obtained

## 5.4.3 Context

This section contains context related definitions. Most of these definitions computationally specify how to construct the definitional elements from the declarations in the previous section.

**Definition** (*Context Frame of a Semantic Type*)
Let $\tau$ be a semantic type, the context frame of $\tau$, $M(\tau)$, is defined as a set as follows:
$M(basic) = \emptyset$.
$i=1..n, m_i \in M(\tau) \leftarrow modifiers(\tau,[m_1,\ldots,m_n])$.
$m \in M(\tau) \leftarrow is\_a(\tau,\tau_s), m \in M(\tau_s)$.

*Informally*, the context frame of a semantic type is recursively defined as the union of its modifier set and its parent context frame. The context frame of semantic type basic constitutes the base case, and has empty context frame.
*Intuitively*, the context frame of a semantic type corresponds to its logical[33] modifier set.

***Example:***

- The context frame of semantic type price is {type, coverage, currency} (see Figure 5.5)

**Definition** (*Extensional Context of a Semantic Object*)
Let t be a semantic object of type $\tau$, the extensional context c of object t, $C_E(t, c)$, is defined as a set as follows:
$C_E(t,basic) = \emptyset$.
$\{m,t_v\} \in C_E(t,c) \leftarrow m \in M(\tau), modifier(\tau, t, m, c, t'), value(t',c,t_v)$.
$\{m,t_v\} \in C_E(t,c) \leftarrow is\_a(c,c_s), \{m,t_v\} \in C_E(t,c_s), \{m,t'\} \notin C_E(t,c)$.

[32] While this may seem like a counter intuitive attribute function, each data cell is uniquely represented in . With that in mind, it becomes reasonable to see a price data cell referring to its product.
[33] Referring not only to physical, but also to inferable items.

.

*Informally*, given a context c the extensional context of a semantic object is the set of modifier, value tuples in that context. Modifiers are obtained from the context frame of its semantic type. Modifier values are derived from modifier declarations for the semantic object in context c, or any of its supertypes, the sub context values overriding the parent ones.

*Intuitively*, the extensional context of a semantic type is its context frame appended with modifier values.

***Example:***
Extensional context for the objects of semantic type Price in yahoo context is as follows:
{<type, "nominal" >, <coverage, "one-way">, <currency, "USD">}

**Definition** (*Intensional Context of a Semantic Object*)
Let t be a semantic object of type $\tau$, t′ be a semantic object, the intensional context c of object t, $C_I(t, c)$, is defined as a set as follows:
$C_I(t,basic) = \emptyset$.
(modifier(_[34], t, m, c, t') $\leftarrow L_1, \ldots, L_n) \in C_I(t,c) \leftarrow m \in M(\tau)$.
(modifier(_, t, m, $c_s$, t') $\leftarrow L_1, \ldots, L_n) \in C_I(t,c) \leftarrow$ is_a(c,$c_s$),
(modifier(_, t, m, $c_s$, t') $\leftarrow L_1, \ldots, L_n) \in C_I(t,c_s)$,
(modifier(_, t, m, c, t') $\leftarrow L_1, \ldots, L_n) \notin C_I(t,c)$.

*Informally*, intensional context c of a semantic object t is recursively defined as the union of its modifier atoms corresponding to the context frame of its semantic type, and the intensional context $c_s$ of t, where $c_s$ is a supert-type of context c. Sub context modifier atoms override super type modifier atoms.

*Intuitively*, the intensional context of a semantic object is the logical set of modifier atom declarations.

**Definition** (*Semantic Object Set of a Primitive Relation*)
Let r be a relation in source s, let $t_i'$ be the elevation[35] of $t_i$, the semantic object set of this relation $S_O(r)$ is defined as follows:
$t_i' \in S_O(r) \leftarrow$ relation(s, r, _, S), $[t_i,\tau_i] \in$ S.

*Informally*, semantic object set of a relation are those obtained during elevation. See mapping definitions for more detail.
*Intuitively*, the semantic object set of a primitive relation is the set of semantic objects corresponding to columns in a relation.

***Example:*** The semantic object set of the cityairport relation from the airfare example is {City′,Airport′} as shown in Figure 5.5.

**Definition** (*Context Identifier Set of a Primitive Relation*)
The context identifier set of a primitive relation r, CI(r), is the set of context identifiers used in elevating r to semantic relation r′.

***Example:*** The context identifiers of the cityairport relation from the airfare example is {c_us} as shown in Figure 5.5.

[34] _ means "any" in logic programming.
[35] Refer to mapping definitions.

.

**Definition** (*Context of a Relation*)
Extensional and intensional contexts of a relation r, denoted by $C_E(r)$, $C_I(r)$ are defined as follows:
$\{t,c,m,v\} \in C_E(r) \leftarrow t \in S_O(r), c \in CI(r), \{m,v\} \in C_E(t, c)$.
$C_I(r) \supseteq C_I(t, c) \leftarrow t \in S_O(r), c \in CI(r), C_I(t, c)$.
where $S_O(r)$ is the semantic object set of primitive relation r.

*Informally*, the extensional/intensional context of a relation r is the union of its extensional/intensional contexts defined for the semantic objects and context identifiers of that relation.
*Intuitively*, extensional/intensional context of a relation is the union of the extensional/intensional contexts of its semantic types.

***Example:*** The extensional context of cityAirport relation is
**{**< Airport′, c_us, format, "airportname">}
The intensional context of cityAirport relation is
{modifier(airport, Object, format, c_ us, Modifier) ←
cste(airportName, Modifier, c_ us, "airportname").}

**Definition** (*Context of a Source*)
Extensional and intensional context c of a source s, denoted by $C_E(s)$, $C_I(s)$ is defined as follows:
$C_E(s) \supseteq C_E(r) \leftarrow$ relation(s, r, _).
$C_I(s) \supseteq C_I(r) \leftarrow$ relation(s, r, _).
This is simply the union of contexts of relations that are included in a source.

**Definition** (*Context Referred by Identifier*)
Extensional and intensional context referred by an identifier c, $C_E(c)$, $C_I(c)$ is defined as follows:
$\{\tau,m,v\} \in C_E(c) \leftarrow \{m,v\} \in C_E(t, c)$.
$C_I(c) \supseteq C_I(t, c) \leftarrow C_I(t, c)$.
where t is of semantic type $\tau$.

*Informally*, the extensional context referred by a context identifier c is the set of semantic type, modifier, and modifier value triples where the modifier and its value are obtained from the extensional context definitions referring to context identifier c. Similarly, intensional context referred by a context identifier c subsumes all the intensional context elements referring to context identifier c.

*Intuitively*, context referred by identifiers is the set of all context definitions with the same (logical) identifier.

***Example:*** Extensional context of c_us is as follows:

- **{**< airport, format, "airportname">, < moneyAmount, currency, "USD">, < currency, format, "3char">}

Intensional contexts from the airfare example are shown in Figure 5.4.

**Definition** (*Context Frame of an Ontology*)
Context frame of an ontology O, is a set defined over the semantic types S of an ontology as follows:

$\{\tau, C(\tau)\} \in C(O) \leftarrow C(\tau), C(\tau) \neq \varnothing, \tau \in S.$

*Intuitively*, context frame of an ontology is a collection of all of non-empty context frames of its semantic types.

***Example:*** Context frame of the airfare ontology is as follows:

{

{moneyAmount, {currency}},

{currency, {format}},

{airport, {format}},

{price, {currency, coverage, type}},

{date, {format}}

}

**Definition** (*Extensional Context of an Ontology*)
Let S be the set of sources that subscribe[36] to an ontology O, the extensional context of an ontology is defined as follows:

$C_E(O) \supseteq C_E(s) \leftarrow C_E(s), s \in S.$

*Informally*, this is equivalent to the union of extensional source contexts that subscribe to the ontology.

## 5.4.4 Sources and Constraints

This section contains definitions relevant to sources and their constraints.

**Definition** (*Relations of a source*)
Relation set of a source $s_i$, $R(s_i)$ is defined as

$$\text{relation}(s_i, R, \_, S) \in R(s_i) \leftarrow \text{relation}(s_i, R, \_, S).$$

***Example:***

R(yahooDB) = {relation(yahooDB, yahoo, I, [[…,…]….]) , relation(yahooDB, cityAirport, i, [[city,string],[airport, string]])}

**Definition (***Integrity Constraints of a relation*)
Integrity constraints of a relation r, SC(r), is a collection of

- key constraints that express the keys of a primitive relation
- foreign key constraints constraining the links between relations

[36] Refer to mapping definitions.

*c_us context*

modifier(moneyAmount, Object, currency, c_us, Modifier) ←

Modifier = skolem(currency, “USD” , c_us, 1, constant("USD")).

modifier(currency, Object, format, c_us, Modifier) ←

Modifier = skolem(currencySymbol, “3char” , c_us, 1, constant("3char")).

modifier(airport, Object, format, c_ us, Modifier) ←

Modifier = skolem(airportName, “airportname” , c_ us, 1, constant("airportname")).

*c_uk context*

modifier(moneyAmount, Object, currency, c_us, Modifier) ←

Modifier = skolem(currency, “£” , c_us, 1, constant("£")).

modifier(currency, Object, format, c_uk, Modifier) ←

Modifier = skolem(currencySymbol, “1char” , c_uk, 1, constant("1char")).

*c_yahoo context*

modifier(price, Object, type, c_yahoo, Modifier) ←

Modifier = skolem(priceType, “nominal” , c_yahoo, 1, constant("nominal")).

modifier(price, Object, coverage, c_yahoo, Modifier) ←

Modifier = skolem(priceType, “oneway” , c_yahoo, 1, constant("oneway")).

modifier(airport, Object, format, c_yahoo, Modifier) ←

Modifier = skolem(airportName, “3ltrCode” , c_yahoo, 1, constant("3ltrCode")).

modifier(date, Object, dateformat, c_yahoo, Modifier) ←

Modifier = skolem(dateType, “American” , c_yahoo, 1, constant("American")).

*c_user context*

modifier(price, Object, type, c_user, Modifier) ←

Modifier = skolem(priceType, “final” , c_user, 1, constant("final")).

modifier(price, Object, coverage, c_user, Modifier) ←

Modifier = skolem(priceType, “roundtrip” , c_user, 1, constant("roundtrip")).

modifier(date, Object, dateformat, c_user, Modifier) ←

Modifier = skolem(dateType, “European” , c_user, 1, constant("European")).

modifier(airport, Object, format, c_ user, Modifier) ←

Modifier = skolem(airportName, “cityname” , c_ user, 1, constant("cityname")).

Figure 5.4 Intensional Context Declarations for the Airfare Example

- general constraints that may involve semantic conflicts

These constraints are expressed as constraint rules.

***Example:***

- Key constraint for the crcyconvert relation

crcyconvert(FromCur, ToCur, $ExchangeRate_1$, Date), crcyconvert(FromCur, ToCur, $ExchangeRate_2$, Date) → $ExchangeRate_1 = ExchangeRate_2$.

- General constraint for the yahoo relation

yahoo(ID,Airline, Price, Tax, DepDate, ArrDate, DepCity, CxnCountry, ArrCity),
DepDate > ArrDate → invalid.

**Definition (***Integrity Constraints of a source*)
Integrity constraints of a source s, SC(s), is defined as
$SC(s) \supseteq SC(r) \leftarrow relation(s, r, S) \in R(s)$

*Informally*, this is the union of integrity constraints of each relation that belongs to source s.

## 5.4.5 Mappings (Elevation Axioms)

This section contains definitions related to mappings between the sources and the shared ontology.

**Definition** (*Mappings or Elevations*)

- Given that $t_i$ is of primitive type $\tau_i$ in relation $r(t_1,\ldots, t_n)$, which is in context c, is elevated to a semantic object $t'_i$ of type $\tau'_i$ with the following skolem function:

$$t'_i = skolem(\tau'_i, t_i, c\_i, r(t_1,\ldots, t_n))$$

- Given that $t_1,\ldots,t_n$ are of primitive types $\tau_1,\ldots,\tau_n$, and $t'_1, \ldots,t'_n$ are of semantic types $\tau'_1, \ldots,\tau'_n$ from ontology O, then the source $r_i(t_1,\ldots, t_n)$ in context c is said to elevate to the semantic relation $r_i'(t'_1, \ldots,t'_n)$, if $\forall j = 1..n$, $t_j$ elevates to $t'_j$.
- Elevation of $r_i$, $E(r_i)$, is equal to $\{ r_i'(t'_1, \ldots,t'_n) \leftarrow r_i(t_1,\ldots, t_n), t_1=\ldots,t'_j = skolem(\tau'_j, t_j, c\_j, r_i(t_1,\ldots, t_n)),\ldots\}$
- $r_i$ is said to subscribe to ontology O.

*Informally*, each semantic object corresponds to a cell in primitive relations. Because the cell is being disintegrated from its tuple, skolem[37] function is used to uniquely identify cells with the addition of context and semantic type mapping. Skolem function can also be thought of as an oid.

*Intuitively*, mappings specify how primitive relations are elevated to semantic relations.

***Example:***

- Elevation of a primitive relation to a semantic relation:

$r'_1(t'_1,t'_2) \leftarrow$ $r1(t_1, t_2)$, $t'_1 = skolem(product, t_1, c\_r_{1,} 1, r(t_1, t_2))$,
$t'_2 = skolem(price, t_2, c\_r_{1,} 2, r(t_1, t_2))$.

[37] Skolem functions are used to convert existentially qualified variables into a universal qualified state.

- Elevation rules between sources and the ontology for the airfare example are shown below in Figure 5.5.

**Definition** (*Elevations of a Source*)
Elevation set of a source $s_i$, $E(s_i)$, is defined as follows:
$e \in E(s_i) \leftarrow e \in E(r_i), r_i \in R(s_i)$.

```
yahoo′(ID′,Airline′, Price′, Tax′, DepDate′, ArrDate′, DepCity′, CxnCountry′, ArrCity′) ←
        yahoo(ID,Airline,Price, Tax, DepDate, ArrDate,DepCity,CxnCountry, ArrCity),
        ID′ = skolem(flightID, ID, c_yahoo, 1, yahoo(ID,…, ArrCity)),
        Airline′=skolem(airline, Airline, c_yahoo, 2, yahoo(ID,Airline,…, ArrCity)),
        Price′=skolem(price, Price, c_yahoo, 3, yahoo(ID,Airline,Price,…, ArrCity)),
        Tax′=skolem(tax, Tax, c_yahoo, 4, yahoo(ID,…,Price,Tax,…, ArrCity)),
        DepDate′=skolem(date, DepDate, c_yahoo, 5, yahoo(ID,…,Tax, DepDate,…, ArrCity)),
        ArrDate′=skolem(date, ArrDate, c_yahoo, 6, yahoo(ID,… ,DepDate, ArrDate,…, ArrCity)),
        DepCity′=skolem(airport, DepCity, c_yahoo, 7, yahoo(ID,…,ArrDate, DepCity,…, ArrCity)),
        CxnCountry′=skolem(country, CxnCountry, c_yahoo, 8, yahoo(ID,…,CxnCountry, ArrCity)),
        ArrCity′=skolem(airport, ArrCity, c_yahoo, 9, yahoo(ID,…,CxnCountry, ArrCity)).

crcyconvert′(FromCur′, ToCur′, ExchangeRate′, Date′) ←
        crcyconvert(FromCur, ToCur, ExchangeRate, Date),
        FromCur′ = skolem(currency, FromCur, c_yahoo, 1, crcyconvert(FromCur, …, Date)),
        ToCur′ = skolem(currency, ToCur, c_yahoo, 2, crcyconvert(FromCur, ToCur, …, Date)),
        ExchangeRate′ = skolem(basic, ExchangeRate, c_yahoo, 3, crcyconvert(…, ExchangeRate,…)),
        Date′ = skolem(date, Date, c_yahoo, 4, crcyconvert(FromCur, ToCur, …, Date)).

cityAirport′(City′, Airport′) ←
        cityAirport(City, Airport),
        City′ = skolem(basic, City, c_us, 1, cityAirport(City, Airport)),
        Airport′ = skolem(airport, Airport, c_us, 2, cityAirport(City, Airport)).

visaFees′(VisaType′, VisaFee′) ←
        visaFees(VisaType, VisaFee),
        VisaType′ = skolem(basic, VisaType, c_uk, 1, visaFees(VisaType, VisaFee)),
        VisaFee′ = skolem(visaFee, VisaFee, c_uk, 2, visaFees(VisaType, VisaFee)).
```

Figure 5.5 Elevation Rules for AirFare Example

## 5.4.6 Ontology

**Definition** (*Attributes of a Type*)
Let $\tau$ be a semantic type, the attributes of $\tau$, $A(\tau)$, is defined as a set as follows:
$A(basic) = \varnothing$.
$i=1\ldots m, a_i \in A(\tau) \leftarrow attribute(\tau, [a_1, \ldots, a_m])$.
$a \in A(\tau) \leftarrow is\_a(\tau,\tau_s), a \in A(\tau_s)$.

*Informally*, attributes of a type are the union of its direct attributes, and any of its supertypes.

**Definition** (*Ontology in ECOIN framework*)

The ontology O in ECOIN framework is $T \cup C \cup H \cup A \cup M$ , where

- T is a set of semantic types $\cup$ {basic}
- C is a set of context identifiers $\cup$ {basic}
- H is a set of clauses defining the sub-type relationships
  - between semantic types in T, and
  - between context identifiers in C
- A is the set of declarations of $a \in A(\tau)$, $\tau \in T$
- M is the set of declarations of $m \in M(\tau)$, $\tau \in T$

***Example:***

- The ontology graphically depicted in Figure 5.6 is expressed in Figure 5.7.

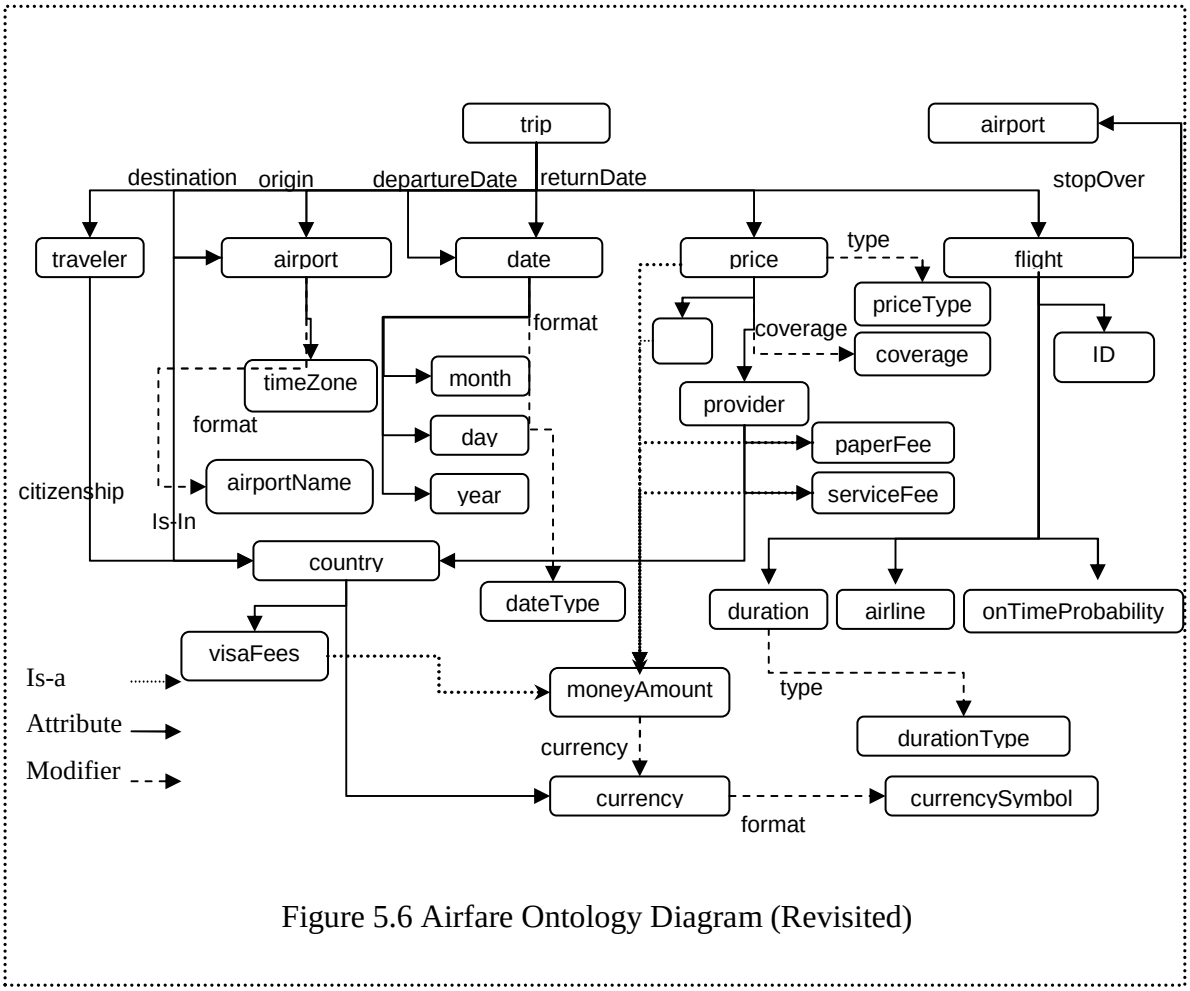



Figure 5.6 Airfare Ontology Diagram (Revisited)

T

semantic_types([trip, airport, traveler, airport, date, price, flight, timeZone, priceType, coverage, flightID, month, day, year, airportName, provider, paperFee, serviceFee, country, dateType, duration, airline, onTimeProbability, visaFees, moneyAmount, currency, durationType, currencySymbol, tax, basic]).

C

contexts([c_yahoo, c_user, c_uk, c_us]).

H

is_a(price, moneyAmount).
is_a(paperFee, moneyAmount).
is_a(serviceFee, moneyAmount).
is_a(visaFees, moneyAmount).

is_a(c_yahoo, c_us).
is_a(c_user, c_us).

A

attribute(trip, destination, airport).
attribute(trip, origin, airport).
attribute(trip, traveler, traveler).
attribute(trip, departuredate, date).
attribute(trip, returndate, date).
attribute(trip, price, price).
attribute(trip, flight, flight).
attribute(trip, isin, country).
attribute(flight, stopOver, airport).
attribute(flight, flightID, flightID).
attribute(flight, duration, duration).
attribute(flight, airline, airline).
attribute(flight, onTimeP, onTimeProbability).
attribute(country, officialCurrency, currency).
attribute(country, visaFees, visaFees).
attribute(traveler, citizenship, country).
attribute(provider, isin, country).
attribute(provider, paperFee, paperFee).
attribute(provider, serviceFee, serviceFee).
attribute(date, day, day).
attribute(date, month, month).
attribute(date, year, year).
attribute(price, tax, tax)

M

modifier(moneyAmount, currency, currency).
modifier(currency, format, currencySymbol).
modifier(duration, type, durationType).
modifier(date, dateFormat, dateType).
modifier(price, type, priceType).
modifier(price, coverage, coverage).
modifier(airport, format, airportName).

Figure 5.7 ECOIN Airfare Ontology

## Conversion Functions

**Definition (***Conversion function*)
A conversion function for an ontology O is a mapping $f$: $S \times P \times M \times C \times M_v(m) \times M_v(m) \rightarrow P$ where

- S is a domain of semantic objects in O
- P is a domain of primitive objects
- M is a domain of modifier names in O
- C is a domain of context identifiers in O
- $M_v(m)$ is a domain of modifier values for modifier m defined as follows:
  $v \in M_v(m) \leftarrow <t, c, m, v> \in C_E(O)$

**Example:** The conversion function for modifier currency is as follows:
$t'_p = f\ (t, t_p, \text{currency}, c, MV_s, MV_t) =$
$t_p * \text{value}(\text{exchange}(\text{value}^{-1}(MV_s, c), \text{value}^{-1}(MV_t, c)), c)$
where exchange and value are external functions.
This conversion function converts the value $t_p$ of semantic object t to value $t'_p$ given the source and target modifier values $MV_s$ and $MV_t$ (e.g. USD and EUR).

**Definition** (*Commutative Conversion Function*)
A conversion function is called commutative if

$$\mathbf{v} = f\ (x, \mathbf{y}, z, p, \mathbf{q}, \mathbf{r}) \leftrightarrow \mathbf{y} = f\ (x, \mathbf{v}, z, p, \mathbf{r}, \mathbf{q})$$

*Intuitively*, commutative conversion functions are those that are symmetric with respect to {source value, source modifier value} and {target value, target modifier value} pairs.

***Example:*** The conversion function for modifier currency is commutative if
exchange(x,y) = 1 / exchange(y,x) --which is true for currency conversion--, as derived below:
$v = y * \text{value}(\text{exchange}(\text{value}^{-1}(q,p), \text{value}^{-1}(r,p)), p)$
$y = v * \text{value}(\text{exchange}(\text{value}^{-1}(r,p), \text{value}^{-1}(q,p)), p)$
$\text{value}(\text{exchange}(\text{value}^{-1}(q,p), \text{value}^{-1}(r,p)), p) = 1/ \text{value}(\text{exchange}(\text{value}^{-1}(r,p), \text{value}^{-1}(q,p)), p)$
$\text{exchange}(\text{value}^{-1}(q,p), \text{value}^{-1}(r,p)) = 1/ \text{exchange}(\text{value}^{-1}(r,p), \text{value}^{-1}(q,p)))$

**Definition** (*Representation of a Conversion Function*)
A conversion function $f$ (x, y, z, p, q, r) is represented as

$$\text{cvt}\ (\text{property}, x, y, z, p, q, r) \leftarrow L_1, \ldots, L_n.$$

where $L_1, \ldots, L_n$ are atoms or user defined literals, property is commutative if $f$ is.

***Example:*** The conversion function for modifier currency is represented as:

$\text{cvt}\ (\text{commutative}, t, t_p, \text{currency}, MV_s, MV_t, t_p') \leftarrow$
$\text{exchangeRate}'(\text{Currency}_1, \text{Currency}_2, \text{Rate}),$
$\text{value}(\text{Currency}_1, c, MV_s),$
$\text{value}(\text{Currency}_2, c, MV_t),$
$\text{value}(\text{Rate}, c, \text{Rate}_p),$

.

multiply($t_p$ ,$Rate_p$, $t_p'$).

In the above example, conversion function is specified using the semantic relation exchangeRate′, value functions, and an arithmetic operator multiply.

**Definition** (*Conversion Functions of an Ontology*)
Conversion functions of an ontology, CF, is defined as follows:
$f$ (x, y, m, p, q, r) $\in$ CF $\leftarrow$ <$\tau$, C($\tau$)>$\in$C(O), <m, $\tau$'> $\in$ C($\tau$)

*Informally,* this corresponds to the set of conversion functions defined for all modifiers in an ontology.

## 5.4.7 ECOIN Framework

**Definition** The ECOIN framework is S $\cup$ O $\cup$ E $\cup$ C $\cup$ CF $\cup$ CS where

- S , the source set, is $R(s_1) \cup R(s_2) \ldots \cup R(s_k)$ where $s_i$ is a source symbol.
- O is an ontology,
- E, the elevation set, is $E(s_1) \cup E(s_2) \ldots \cup E(s_k)$
- C, the context set, is $C_I(c_1) \cup C_I(c_2) \ldots \cup C_I(c_n)$ where $c_i$ is a context symbol.
- CS, the conversion functions,
- IC, the set of integrity constraints, is $IC(s_1) \cup IC(s_2) \ldots \cup IC(s_k)$

**Definition** An ECOIN *application* is an instance of the ECOIN framework.

This completes the formal specification of knowledge representation in ECOIN. Applications using the representation framework detailed in this section, can take advantage of the reasoning facilities we provide to achieve semantic interoperability among heterogeneous and autonomous sources. In the next section, we will focus on the reasoning framework by considering query answering in ECOIN, with a particular emphasis on equational ontology constraints.

# Chapter 6

# Query Answering in ECOIN

We have grouped data heterogeneities into three categories in Chapter 3: contextual, ontological, and temporal. Query answering techniques in COIN framework, as briefly explained Chapter 4, successfully handle a subset of contextual heterogeneities using abductive reasoning. Equational ontological heterogeneities (EOC), on the other hand, even when they are transformed and represented as contextual heterogeneities, require a reasoning technique that intertwines abductive reasoning with symbolic equation solving.

In this chapter, we describe an extended reasoning approach that encodes symbolic equation solving through the use of *constraint handling rules* (CHR) [Frühwirth 98], a high-level language extension of *constraint logic programming (CLP)* for writing custom constraint solvers. This extension, coupled with abductive reasoning provides an elegant and powerful solution to the problem of detecting and resolving EOC.

In this section, we, first, describe EOC in more detail by providing examples from financial information systems. Then, we explain ECOIN approach to resolving EOC, which is based on *abductive constraint logic programming*, a combination of abduction and constraint solving techniques [Kakas 00].

## 6.1 Equational Ontological Conflicts

In Chapter 3, we referred to a financial case study conducted in Primark and summarized our findings concerning different types of heterogeneities in financial information systems. In that study, we found that many data items are derived from other simpler data items. For example, *Price Earnings Ratio* is calculated by dividing price per share by earnings per share. However, this definition is subject to multiple interpretations, as it does not specify whether the earnings are "*trailing*"[38] or "*forward*"[39], or more importantly what is included in the earnings. In fact, when we collected *Price Earnings Ratios* for a specific company, Daimler-Benz, from several financial sources on the same day the numbers differed significantly, because of the differences in the interpretation of *earnings* (see Table 6.1.) A closer examination reveals that these

[38] Trailing earnings are earnings in the last 12 months.
[39] Forward earnings are based on future earning estimates.

variations are not caused by erroneous reporting, but attributable to definitional differences among data sources.

| SOURCE | P/E RATIO |
|---|---|
| ABC | 11.6 |
| Bloomberg | 5.57 |
| DBC | 19.19 |
| MarketGuide | 7.46 |

Table 6.1 Key Financials for Daimler-Benz (from [Madnick 01]).

Financial concepts such as "Revenues", "Expenses" and "Profits" are ontologically distinct but have interdependences that can be expressed as equations, such as "Profit = Revenues – Expenses." We refer to the heterogeneity in the way data items are calculated from other data items in terms of definitional equations, as *equational ontological conflicts*. Such conflicts in accounting methods are quite widespread not only between different countries, but also within the same country. For example, *The Wall Street Journal* and *S&P* use different methods to calculate the *P/E Ratios* for the *Standard & Poor's 500-stock index*. *The Wall Street Journal* divides the combined market capitalization of the 500 companies currently in the index by their *most recently reported four quarters of earnings*, while S&P updates earnings statistics for the index *just once a quarter* and doesn't revise earnings from previously reported quarters to account for additions or deletions to the index[40].

As long as the context used by each source of financial data is known, there is nothing wrong with a multiplicity of calculation methods – i.e., of equational ontologies. Yet, problems occur once companies' financial numbers, crunched by analysts, enter a vast information food chain, where they are repeated, often without explanation, in hundreds of news sources, and end up being used out of context. This becomes even more challenging when there is the need to combine or compare data obtained from multiple sources with differing contexts.

In ECOIN, equational ontological conflicts are not handled by introducing new types in the ontology and defining equational relationships at the ontological level. Introducing new types is likely to be a time-consuming and difficult process, and is better avoided. Furthermore, in many cases, such an approach would result in an explosion of new ontology types to handle all of the possible variations. We adopt our context-based solution to this problem by making the context of the data items of each source explicit (i.e., how they are derived from other data items) and adjusting their values to different contexts by recalculating them when necessary using the context information – including the definitional equations associated with each context.

Consider the airfare example we described in Chapter 4. Although the ontology corresponding to the airfare example has a single concept called *price*, it assumes different meanings in different contexts as shown in Figure 6.1. Definitional differences between different price elements are expressed by using the *type* modifier. Conversion functions are then used to define the relationships, or in this specific case equations,

[40] Moving Target: What's the P/E Ratio? Well, Depends on What Is Meant by Earnings --- Terms Like `Operating,' `Core,' `Pro Forma' Catch Fire, Leave Investors Muddled --- `Earnings Before Bad Stuff', Jonathan Weil, Wall Street Journal, Aug 21, 2001.

between different modifier values. In conversion libraries, it is enough for new additions simply to establish a connection to the network of conversion functions, and then our system automatically takes care of combining, inverting and simplifying them through the use of Dijkstra's shortest path algorithm and the use of Constraint Handling Rules.

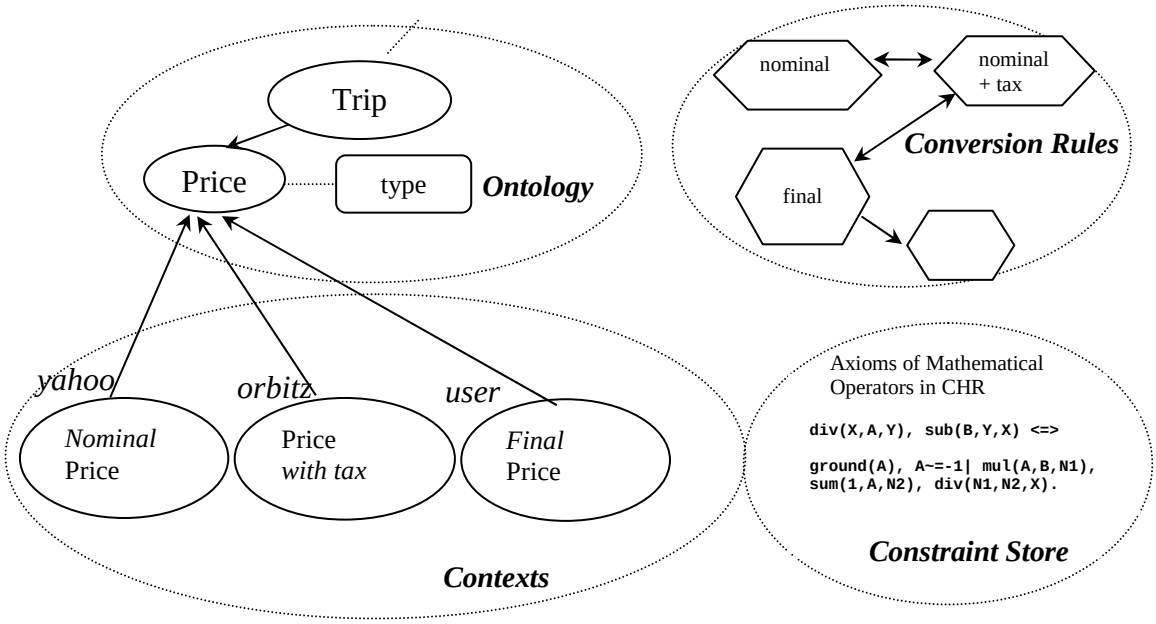



Figure 6.1. Equational Conflict Handling in ECOIN

Below we explain the details of query answering in ECOIN framework by first providing background on abductive constraint logic programming.

# 6.2 Abductive Logic Programming

The notion of *abduction* was first introduced by philosopher Peirce [Peirce 1903] as another form of synthetic inference, deriving the *facts* from *rules* and *results*. This is different both from *deduction*, which derives the *results* from *rules* and *facts*; and *induction*, which derives the *rules* from *results* and *facts*. Abductive derivations are possible explanations that are consistent with the observed facts and the rules. In that sense, abductive explanations are hypothetical, as those explanations may later have to be retracted when new facts are available.

*Abductive logic programming* (ALP) is an extension of a logic programming (LP) to perform abduction. It is increasingly being used in many complex AI problems such as in problems of diagnosis, planning and scheduling, natural language understanding, database updates, and information integration [Goh 97]. In all of these applications the required goals to be solved are seen as observations to be explained by abduction (e.g. a mediated query in information integration is viewed as an explanation of the intended user query).

.

An *abductive logic programming framework* is defined as a triple (T, A, IC) consisting of a logical program T, a set of abducible predicates A, and a set of classical logic formulas IC, called the integrity constraints [Denecker and Kakas 01]. Given a query Q, its abductive explanation is equivalent to finding a set of abducible predicates $\Delta \in A$ such that:

1. $T \cup \Delta \models Q$ (i.e theory and abducibles entails the query)
2. $T \cup \Delta$ satisfies IC
3. $T \cup \Delta$ is consistent

In general, the set of abducibles, $\Delta$, are ground. If non-ground abducibles such as $\Delta = \{\exists X\ a(X)\}$ are allowed the corresponding framework is known as *constructive abduction* [Kakas and Manceralla 93]. The integrity checking of such abducible hypotheses can naturally be understood in constraint logic programming terms, which we explain next.

# 6.3 Constraint Logic Programming

*Constraint logic programming* (CLP) is an extension of ordinary LP, with constraint predicates, that are checked for satisfiability and simplified by means of a constraint solver. Like LP, a CLP program needs to search a database of facts, but it can use constraints to rule out many possible outcomes and prune away large parts of the search tree. In CLP the unification algorithm is augmented by a dedicated solver applying constraint-solving algorithms from other branches of computing such as the simplex method from Operations Research. As a result CLP programs, in general, are more concise and efficient than ordinary LP. Compare for example, the CLP and LP versions[41] of the N queens problem[42] shown in Table 6.2 below:
In CLP, a *constraint* is like any other logic predicate and when called is *posted* to the constraint store. Several types of constraints can be declared using CLP, including:

- Arithmetic Constraints
  e.g. X #\= Y (X is not equal to Y) .
- Membership Constraints
  e.g. `Board[1..N] :: 1..N (=each element of the board array must be an element of {1,…,N}) ,`
- Propositional Constraints
  e.g. *`P #/\ Q`* `(= True if the constraints P and Q are both true)`
- Combinatorial Constraints
  e.g. `all_different(`*`Variables`*`) (= True if the variables do not have identical values)`
- User-Defined Constraints
  e.g. prime number constraints using *constraint handling rules*

```
primes(1) <=> true.
primes(N) <=> N> 1 | M is N – 1, prime(N), prime(M).
prime(I), prime(J) <=> 0 is J mod I | prime(I).
```

[41] Both programs are in Sicstus prolog.
[42] Place N chess queens in a N×N board such that they do not attack each other

.

| CLP Formulation (from Eclipse Manual) | LP formulation (from Craft of Prolog) |
|---|---|
| queens_arrays(N, Board) :-<br>dim(Board, [N]),<br>Board[1..N] :: 1..N,<br><br>( for(I,1,N), param(Board,N) do<br>( for(J,I+1,N), param(Board,I) do<br>Board[I] #\= Board[J],<br>Board[I] #\= Board[J]+J-I,<br>Board[I] #\= Board[J]+I-J<br>)<br>),<br><br>Board =.. [_\|Vars],<br>labeling(Vars). | queens(N, Queens) :-<br>length(Queens, N),<br>board(Queens, Board, 0, N, _, _),<br>queens(Board, 0, Queens).<br>board([], [], N, N, _, _).<br>board([_\|Queens], [Col-Vars\|Board], Col0, N,<br>[_\|VR], VC) :-<br>Col is Col0+1,<br>functor(Vars, f, N),<br>constraints(N, Vars, VR, VC),<br>board(Queens, Board, Col, N, VR, [_\|VC]).<br>constraints(0, _, _, _) :- !.<br>constraints(N, Row, [R\|Rs], [C\|Cs]) :-<br>arg(N, Row, R-C),<br>M is N-1,<br>constraints(M, Row, Rs, Cs).<br>queens([], _, []).<br>queens([C\|Cs], Row0, [Row-Col\|Solution]) :-<br>Row is Row0+1,<br>select(Col-Vars, [C\|Cs], Board),<br>arg(Row, Vars, Row-Row),<br>queens(Board, Row, Solution).<br>select(X, [X\|R], R).<br>select(X, [H\|T], [H\|R]) :-<br>select(X, T, R). |

Table 6.2 N Queens formulation in CLP and LP

*Enumeration predicates* check satisfiability of the constraints by instantiating variables through specific algorithms, such as *branch and bound* in the case of `maximize(Goal,X)` (i.e. find the solution of Goal that maximizes X) or the more general `labeling(Options, Variables)` in which a set of search options can be specified. In the N-Queens example shown in Table 6.1, the default labeling is used, which instantiates variables starting from the smallest element in its feasible set.

A *constraint solver* supports some of the basic operations such as *satisfaction, simplification, propagation, normalization, entailment, and optimization.* While constraint solvers were in the beginning black box systems, languages such as *constraint handling rules* (CHR) now allow users write their own constraint solvers in a high level language [Frühwirth 98]. It has been used to construct a wide range of solvers including terminological and temporal reasoning. Next we provide more details on CHR.

## 6.3.1 Constraint Handling Rules

The theory of constraint handling rules (CHR), including its implementation, was proposed by Frühwirth. In this section we provide a summary of the syntax and semantics

of CHR and some important theoretical results concerning its soundness and completeness by referring to [Frühwirth 98].

### Syntax of CHR

**Definition:** (CHR Program)

A CHR program is a finite set of CHR. There are three kinds of CHR.

A *simplification* CHR is of the form:

$H_{1,}..., H_i <=> G_1,\ldots, G_j \mid B_1, \ldots, B_k$,

a *propagation* CHR is of the form

$H_{1,}..., H_i ==> G_1,\ldots, G_j \mid B_1, \ldots, B_k$,

a *simpagation* CHR is of the form

$H_{1,}..., H_l \setminus H_{1+1,}..., H_i <=> G_1,\ldots, G_j \mid B_1, \ldots, B_k$,

with $i > 0$, $j \geq 0$; $k \geq 0$, $l > 0$ and where the multi-head $H_{1,}..., H_i$ is a nonempty sequence of CHR constraints, the guard $G_1,\ldots, G_j$ is a sequence of built-in constraints, and the body $B_1, \ldots, B_k$ is a sequence of built-in and CHR constraints.

Above, a simpagation rule is an abbreviation for the following simplification rule:

$H_{1,}..., H_l, H_{1+1,}..., H_i <=> G_1,\ldots, G_j \mid H_{1,}..., H_l, B_1, \ldots, B_k$.

therefore will be dealt as simplification from now on.

Below is an example set of CHR for simplification, propagation and the use of guards:

reflexivity @ X =< Y <=> X=Y | true.

antisymmetry @ X=<Y, Y=<X <=> X=Y.

transitivity @ X=<Y, Y=<Z ==> X=<Z.

The first rule replaces X=<Y with true (an empty sequence) provided that X=Y. Thus whenever the constraint solver encounters the constraint X=<X it is simplified to true. The second rule means that whenever we find X=<Y, as well as Y=<X in the current constraint we can replace it with the logically equivalent X=Y. Finally, the transitivity adds the new redundant constraint, X=<Z, to the store whenever it encounters both X=<Y and Y=<Z in the current constraint. This new constraint although redundant may activate other rules in the constraint store and achieve useful simplifications.

## 6.3.2 Declarative Semantics of CHR

One of the distinguishing features of CHR compared to an LP language, such as Prolog is the allowance of multiple heads in the clauses. While joint reductions of multiple atoms are analogous to production rules of expert system languages such as OPS5, the similarity is merely syntactical. Rules in production rules like systems involve non-monotonicity, e.g. state changes caused by actions or method calls, as opposed to declarative constraint solving.

Declarative meaning of a CHR program is defined as follows:

**Definition** (Declarative meaning)

Declaratively, a simplification CHR is a logical equivalence if the guard is satisfied:

$\forall x (\exists y (G_1 \wedge \ldots \wedge G_j)) \rightarrow (H_1 \wedge \ldots \wedge H_i \leftrightarrow \exists z (B_1 \wedge \ldots \wedge B_k))$

A propagation CHR is an implication if the guard is satisfied:

$\forall x (\exists y (G_1 \wedge \ldots \wedge G_j)) \rightarrow (H_1 \wedge \ldots \wedge H_i \rightarrow \exists z (B_1 \wedge \ldots \wedge B_k))$

where x denote the sequence of global variables occurring in the head atoms $H_1, \ldots, H_i$, y (z) are the local variables occurring in the guard $G_1,\ldots, G_j$ (body $B_1, \ldots, B_k$) of a rule.

For example the reflexivity constraint:

reflexivity @ X =< Y <=> X=Y | true.
is equivalent to:
$\forall X,Y\ (X=Y) \rightarrow (X =< Y \leftrightarrow true)$.

## 6.3.3 Operational Semantics of CHR

Operationally CHR programs can be thought of as a state transition system with the state defined as:

**Definition**: (State)
A state is an annotated tuple $<F, E, D>_v$
where F is a conjunction of CHR and built-in constraints (e.g. =, true, false) constituting the goal store (a.k.a the query), E is a conjunction of CHR constraints, and D is a conjunction of built-in constraints called the constraint stores (both), and v is a sequence of variables.

*Initial state* consists of a goal F and empty constraint stores. $<F, true, true>_v$, *final state* is either of the form $<F,E,false>_v$ (called failed) or $<true,E,D>_v$ (called successful answer) with no computation step possible anymore.

In the transition logic, F are the constraints that remain to be solved, and D and E are the constraints that have been accumulated and simplified so far. The aim of the computation is to arrive at a state that contains no more goals. Of the four transition types, one solves built-in constraints, one introduces CHR constraints into their store, and the remaining two apply simplification and propagation CHRs. All transitions leave the annotation v unchanged.

**Definition**: (Transitions)
Let P be a CHR program for the CHR constraints and CT[43] be a built-in constraint theory which determines the meaning of built-in constraints. The transition relation $\rightarrowtail$ for CHR is as follows. All variables in states stand for conjunctions of constraints. x denotes the program variables occurring in the multi head H.

**Solve:** (Updates the constraint store D with a new constraint C from the goal store.)
$<C \wedge F, E, D>_v \rightarrowtail <F, E, D'>_v$
if C is a built-in constraint and CT $\models (C \wedge D) \leftrightarrow D'$

**Introduce:** (Transports a CHR constraint H from the goal store into the CHR constraint store)
$<H \wedge F, E, D>_v \rightarrowtail <F, H \wedge E, D'>_v$
if H is a CHR constraint

**Simplify:** (A simplification rule (H <=> G | B) applying to a CHR constraint H′ removes H′ from the CHR constraints store, adds B to the goal store and adds the equation H=H′ to the built-in constraint store)
$<F, H' \wedge E, D>_v \rightarrowtail <B \wedge F, E, H = H' \wedge D>_v$
if (H <=> G | B) in P and CT $\models D \rightarrow \exists x(H=H' \wedge G)$

**Propagate:** (A propagation rule (H ==> G | B) applying to a CHR constraint H′ adds B to the goal store and adds the equation H=H′ to the built-in constraint store)
$<F, H' \wedge E, D>_v \rightarrowtail <B \wedge F, H' \wedge E, H = H' \wedge D>_v$
if (H ==> G | B) in P and CT $\models D \rightarrow \exists x(H=H' \wedge G)$

[43] at least including {=, true, false}

.

An example transition simulation concerning the reflexivity, antisymmetry, and transitivity CHR rules with the goal $A \leq B \wedge C \leq A \wedge B \leq C$ is shown below (v=[A,B,C] will not be shown):

| Transition | State |
|---|---|
| Initial | $< A \leq B \wedge C \leq A \wedge B \leq C$, true, true> |
| Introduce (x3) | <true, $A \leq B \wedge C \leq A \wedge B \leq C$,true> |
| Propagate (Transitivity) | <$C \leq B$ , $A \leq B \wedge C \leq A \wedge B \leq C$,true> |
| Introduce | <true, $A \leq B \wedge C \leq A \wedge B \leq C \wedge C \leq B$ ,true> |
| Simplify (Antisymmetry) | <B=C, $A \leq B \wedge C \leq A$,true> |
| Solve | <true,$A \leq B \wedge C \leq A$, B=C> |
| Simplify (Antisymmetry) | <A=B, true,B=C> |
| Solve | <true,true, $A=B \wedge B=C$> |

Table 6.2 Example transitions in CHR

### 6.3.4 Soundness and Completeness

The soundness and completeness of CHR programs are established for terminating programs, which trivially follow from the following Lemma:

**Lemma** Let P be a CHR program and G be a goal. If C is the logical reading of a state appearing in a computation of G, then

$P, CT \models \forall (C \leftrightarrow G)$

where $\forall F$ denotes the universal closure of a formula F.

**Theorem** (Soundness) Let P be a CHR program and G be a goal. If G has a computation with answer C then P,

$$P,CT \models \forall (C \leftrightarrow G)$$

**Theorem** (Completeness) Let P be a CHR program and G be a goal with at least one finite computation and C be a conjunction of constraints. If $P, CT \models \forall (C \leftrightarrow G)$, then G has a computation with answer $C'$ such that

$$P, CT \models \forall (C \leftrightarrow C')$$

While the soundness results can be extended to negated goals (failed computations), the completeness result can only be extended under special conditions. In ECOIN, we are dealing with CHR rules for solving linear arithmetic equations. In [Frühwirth 99], the termination of constraints for solving linear polynomial equations is established, which implies their soundness and completeness.

## 6.4 Abductive Constraint Logic Programming

The integration of abductive logic programming with constraint logic programming has been pursued based on the view that they can be both understood within the same conceptual framework of hypothetical reasoning. In both frameworks, an answer to a query is constructed from special predicates (i.e. *abducible* predicates in ALP; *constraint* predicates in CLP) which are constrained either by *integrity constraints* in the case of ALP or by means of *a constraint theory* in CLP. For example, the reflexivity, transitivity and antisymmetry constraints shown before can also be seen as integrity constraints for the abducible inequality predicates. ACLP aims to unify the treatment of abducibles and constraints.

.

While in [Kowalski 92] integrity constraints are used both for abducible and constraint predicates, the opposite approach is taken in [Bürchert 94] and abduction without integrity constraints is treated as a special case of constrained resolution. In [Kakas and Michael 95] the hybrid approach is pursued, in which the central notions of the two frameworks are combined, so that abduction and constraint handling cooperate to solve a common goal. Typically, the goal is reduced first by abduction to abducible hypotheses whose integrity checking reduces this further to a set of constraints to be satisfied in CLP. Finally, in [Kakas 00] an extension of this framework is given, which makes it possible to compute abductive solutions by interfacing constraint solving to abduction. In this case, constraint solver not only solves the final constraint store generated by the abductive reduction but also dynamically affects this abductive search for a solution.

From a formal point of view, ACLP can be seen as an extension of the ALP framework that supports constructive abduction allowing the abducible hypotheses to take the o non-ground form of $\exists X$ (A(X), C(X)) where A is a conjunction of abducible atoms and C is a set of constraints defined over the CLP (arithmetic) domain [Kakas et al. 98]. An ACLP framework can be defined as follows:

**Definition** (ACLP Framework)

An abductive CLP (or ACLP) theory is a triple (P, A, IC) where

- P is a constraint logic program.
- A is a set of abducible predicates different from the constraint predicates.
- IC: is a set of closed first order formulae (Integrity Constraints) over the combined language of CLP and P.

Next we explain, how we utilize the ACLP framework in dealing with *equational ontological conflicts*.

# 6.5 Query Answering with ACLP

Queries in ECOIN are handled similar to the COIN as shown in Figure 6.3, with the exception that the ALP framework in COIN is replaced with ACLP framework in ECOIN. Abduction is now supported by constraint solving capabilities of CLP. In particular CHR is used to express a set of arithmetic constraints whose computational meaning is equivalent to a simultaneous symbolic equation solver. The solver can simplify, invert, and combine equations constructed using arithmetic operators addition, subtraction, division, and multiplication. Its capabilities may easily be extended to more complicated operators such as $\int$, $\partial$, $\sqrt{}$ , but we will limit our discussion to four basic arithmetic operators that may be used to define *polynomial equations*. We should note that symbolic equation solving in ECOIN produces *intensional answers* as opposed to *extensional answers* as in systems like MRDSM [Litwin and Vigier 86]. This difference is crucial in our choice of using CHR instead of a generic symbolic equation solver such as Maple or Macsyma. We provide a comparison of ECOIN with MRDSM after explaining the details of our CHR based solver. Before going into the symbolic capabilities of ECOIN, however, we will explain how a naïve user is query is translated into a well defined query and how the ECOIN framework can be transformed to an ACLP framework.

## 6.5.1 Naïve to Well Defined Query Transformation

As shown in Figure 6.2, users formulate their queries in SQL, which is then trivially converted into its naïve Datalog equivalent. This naïve query, however, does not correspond to the intentions of the user; and needs to be converted into a *well-formed* query. Below we show the naïve query corresponding to the airfare example:

answer(Airline,Price) ←

yahoo(I,Airline, Price, T, “01/06/03”, “01/08/03” , “Boston”, C, “Istanbul”).

Context: c_user

The transformation of a naïve query into a well defined query is accomplished with the following definition:

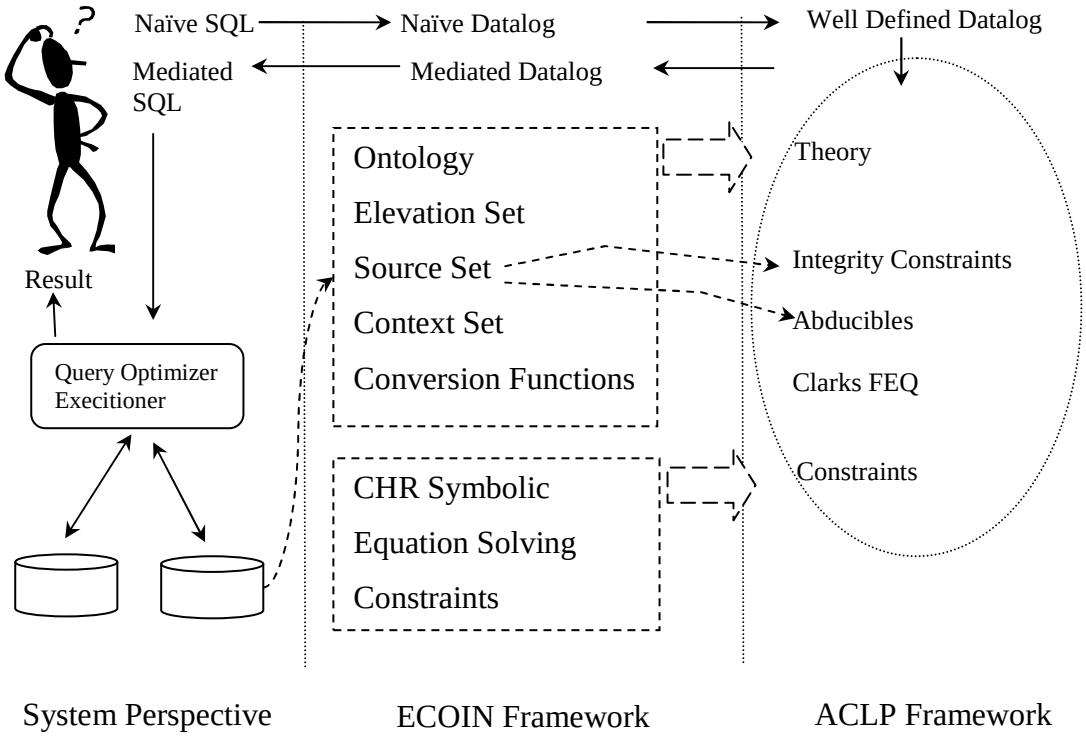



Figure 6.2 ECOIN Query Answering

**Definition** (Naïve to well-defined query transformation)

Let <Q, c> be a naive query in an ECOIN 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 references to extensional relations with the corresponding semantic relations with different variables for data elements; and
- value atoms that map those variables to data elements in extensional relations; for example,

yahoo(I,Airline, Price,T , “01/06/03”, “01/08/03” , “Boston”, C, “Istanbul”).

is replaced by

yahoo′($X_1$, $X_2$, $X_3$, $X_4$, $X_5$, $X_6$, $X_7$, $X_8$, $X_9$).
value($X_1$, c, I),
value(X2, c, Airline),
value(X3, c, Price),
value(X4, c, T),
value(X5, c, “01/06/03”),
value(X6, c, “01/08/03”),
value(X7, c, “Boston”),
value(X8, c, C),
value(X9, c, “Istanbul”).

- Eliminate unnecessary value atoms (i.e. those ones whose third argument is non-ground and not referred by any other query element except the originating relation.

This way the original query
answer(Airline,Price) ←
yahoo(I,Airline, Price, T, “01/06/03”, “01/08/03” , “Boston”, C, “Istanbul”).
Context: c_user
would be transformed into:
answer(Airline,Price) ←
yahoo′($X_1$, $X_2$, $X_3$, $X_4$, $X_5$, $X_6$, $X_7$, $X_8$, $X_9$).
value(X2, c, Airline),
value(X3, c, Price),
value(X5, c, “01/06/03”),
value(X6, c, “01/08/03”),
value(X7, c, “Boston”),
value(X9, c, “Istanbul”).

## 6.5.2 ECOIN to ACLP Transformation

In order to apply ACLP reasoning techniques for *mediating* a well formed user query, we need to establish the relationship between an ECOIN framework and ACLP framework.

**Definition** (Transformation)
An ECOIN framework $\mathcal{F}_E = \{\mathcal{S} \cup \mathcal{O} \cup \mathcal{E} \cup \mathcal{C} \cup \mathcal{CF} \cup \mathcal{CS}\}$ can be mapped to a corresponding ACLP framework $\mathcal{F}_A$ given by $<\mathcal{P}, \mathcal{I}, \mathcal{A}>$ where

- $\mathcal{P}$ is the Datalog$^{neg}$ translation of the set of clauses in $\mathcal{F}_E$ except $\mathcal{CS}$;
- $\mathcal{I}$ consists of the integrity constraints defined in $\mathcal{CS}$, augmented with Clark's Free Equality Axioms, and *symbolic equation solving constraints* in CHR; and
- $\mathcal{A}$ consists of non-ground and ground extensional predicates defined in $\mathcal{S}$, the built-in predicates corresponding to arithmetic and relational (comparison) operators, and the system predicate which provides the interface for system calls.

Note that this transformation is very similar to COIN to ALP transformation, with the exception that $\mathcal{I}$ includes *symbolic equation solving constraints*, which is explained next.

.

## 6.5.3 Symbolic Equation Solving Constraints

Equation solving in ECOIN is necessitated by the existence of conversion functions that convert values between different contexts. As we explained in Chapter 5, ECOIN conversion functions form a network among themselves, and functions denoted as commutative correspond to the bidirectional arrows in Figure 6.3. Before conversion functions between any two contexts are applied, we find the shortest path in the conversion function network by using Dijkstra's shortest path algorithm. Shortest path between two context nodes is calculated by assigning costs to each arc in the network. Currently the costs corresponding to each arc is assumed to be equal, thus the shortest path in the network is the path(s) with the least number of arcs. A random tie breaker is used when more than one path exists.

In order to use the symbolic equation solving capabilities, arithmetic operations in conversion functions are built using basic arithmetic constraint predicates. The definition of a basic arithmetic constraint predicate is as follows:

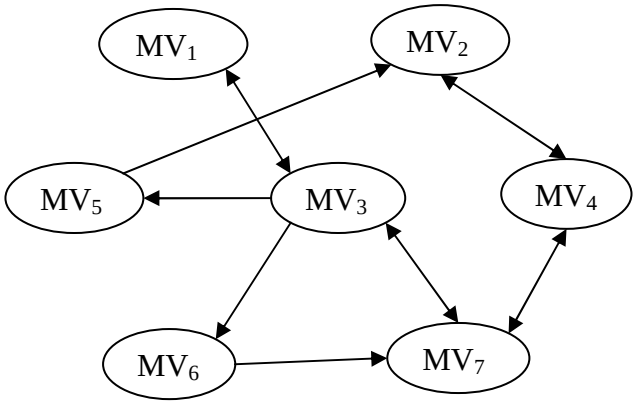



Figure 6.3 A conversion function network for modifier m

**Definition** (Basic Arithmetic Constraint Predicate)
A basic arithmetic constraint predicate is a predicate of arity 3 corresponding to arithmetic operators {+, -, *, \}. The third variable is called the result variable.
**Example:**

- sum($X_1$, $X_2$, $X_3$) is a basic arithmetic constraint predicate corresponding to:

$X_3 = X_1 + X_2$. $X_3$ is the result variable.
Note that by using basic arithmetic predicates, *polynomial conversion functions* of arbitrary complexity can be constructed. For example:

$$f(x, y, z) = x^2y + z/2 - 10$$

can be constructed by the following combination of basic arithmetic predicates:

mul(X,X,$R_1$), mul($R_1$,Y, $R_2$), div(Z,2,$R_3$), sum($R_2$, $R_3$, $R_4$), sub($R_4$,10, F).

.

While applying ACLP in ECOIN, arithmetic predicates along with other predicates is reduced first by constructive abduction to abducible hypotheses, which are further reduced by the application of integrity constraints. Final abducible hypotheses of the non-ground form $\exists X$ (A(X)) are then constrained by the application of integrity and symbolic equation solving constraints.

Before providing a definition of symbolic equation solving constraints, we have to explain the notion of intensional and extensional boundness, which is defined as follows:

**Definition** (Intensional and extensional boundness)

- A variable is *bound* if it is *intensionally* or *extensionally* bound.
- A variable that references a data element in an extensional relation is *intensionally bound*.
- A variable that references a ground atom is called *extensionally bound*.
- A variable X that is functionally dependent on a set of variables S (shown as $S \rightarrow X$) is *intensionally bound* if all elements of S are *bound*.

**Examples:**

- In yahoo($X_1$, $X_2$, $X_3$, $X_4$, $X_5$, $X_6$, $X_7$, $X_8$, $X_9$) each $X_i$ (i=1..9) is intensionally bound.
- In {$X_3$ = 500, $X_4$=50} $X_3$, and $X_4$ are extensionally bound
- In {$X_3$ = 500, $X_4$=50, $X_{10}$ = $X_3$ + $X_4$} $X_{10}$ is intensionally bound for {$X_3$, $X_4$} $\rightarrow X_{10}$ and $X_3$, $X_4$ are bound.

We can now provide a definition of ECOIN symbolic equation solving constraints as follows:

**Definition** (Symbolic Equation Solving Constraint Rules for Arithmetic Predicates)

Symbolic equation solving constraint theory for arithmetic predicates, SCT(A), is a CHR program defined for a set of arithmetic predicates A corresponding to arithmetic operators {+, -, *, /}. SCT(A) reduces a given goal store G, ($p \in G \rightarrow p \in A$), to a constraint store G′ ($p \in G' \rightarrow p \in A$) such that

$\forall p_i (v_{i1}, v_{i2}, v_{i3}) \in G'$; $v_{i1}$, $v_{i2}$, $v_{i3}$ is *bound* if such a reduction exists.

**Example:**

- Suppose {$X_4$,$X_2$} are intensionally bound then SCT({sum,mul,div,sub}) would reduce

G ={sum($X_1$, $X_2$, $X_3$), mul($X_3$, .15, $X_4$) } to G′ ={sub($X_3$, $X_2$, $X_1$), div($X_4$, .15, $X_3$) }
where {$X_4$, .15} $\rightarrow X_3 \wedge$ {$X_3$, $X_2$} $\rightarrow X_1$ establishes feasibility.

Note that the main requirement of a symbolic equation solver is to ensure that all arithmetic predicates are bound. This is established by a set of CHR rules as follows:

**1. If a variable is ground it is also bound**

**Example:**

```
sum(X,Y,Z) ==> ground(X) | bound(X).
sum(X,Y,Z) ==> ground(Y) | bound(Y).
sum(X,Y,Z) ==> ground(Z)  | bound(Z).
```

**2. If a variable is functionally determined by ground values, it is bound and its value can be calculated**

**Example:**

```
div(X,Y,Z) <=> ground(X), ground(Y), nonground(Z), Y ~=0 | Z is X / Y, bound(Z).
```

.

div(X,Y,Z) <=> ground(X), ground(Z), nonground(Y), Z ~=0 , X ~=0 | Y is Z * X, bound(Y).
div(X,Y,Z) <=> ground(Y), ground(Z), nonground(X),Y ~=0 | X is Z * Y, bound(X).
div(0,Y,Z) <=> nonground(Z), Y ~=0 | Z is 0, bound(Z).
div(X,Y,0) <=> nonground(X), Y ~=0 | X is 0, bound(X).
div(X,0,Z) <=> false.

**3. If a result variable is functionally determined by bound values, it is also bound**
**Example:**
sum(X,Y,Z), bound(X), bound(Y) ==> bound(Z).

**4. Identity element constraints**
**Example:**
mul(1,Y,Z) <=> nonground(Z), nonground(Y) | Z=Y.
mul(X,1,Z) <=> nonground(Z), nonground(X) | Z=X.

Note that the assignment here is by reference not by value as in the case of "*is*".

**5. Arithmetic integrity constraints**
**Example:**
sub(A,Y,Z), sub(X,Y,Z) ==> nonground(X), nonground(A) | X = A.
sub(X,Y,Z), sub(X,A,Z) ==> nonground(Y), nonground(A) | Y = A.
sub(X,Y,A), sub(X,Y,Z) ==> nonground(Z), nonground(A) | Z = A.

**6. If a result variable in a predicate is bound simplify that predicate with its inverse**
**Examples:**
sum(X,Y,Z), bound(Z) <=> sub(Z,Y,X), bound(Z).
sub(X,Y,Z), bound(Z) <=> sum(Y,Z,X), bound(Z).
mul(X,Y,Z), bound(Z) <=> Y~=0 | div(Z,Y,X), bound(Z).
div(X,Y,Z), bound(Z) <=> Y~=0 | mul(Y,Z,X), bound(Z).

Note that because of the previous set of constraints all ground predicates are eliminated from the store, while the values of their variables are propagated.

**7. If a result variable in a predicate is bound, and there is another bound variable simplify that predicate with its inverse and declare the remaining variable bound**
**Examples:**
sum(X,Y,Z), bound(X), bound(Z) <=> sub(Z,X,Y), bound(X), bound(Y), bound(Z).
sum(X,Y,Z), bound(Y), bound(Z) <=> sub(Z,Y,X), bound(X), bound(Y), bound(Z).

**8. Interaction constraints**
**Examples:**
mul(X,A,Y), sub(B,Y,X) <=> A~=-1 | div(B,N,X), sum(1,A,N).
mul(X,A,Y), sum(B,Y,X) <=> A~=-1 |div(B,N,X), sub(1,A,N).
div(X,A,Y), sub(B,Y,X) <=> A~=-1 | mul(A,B,N1), sum(1,A,N2), div(N1,N2,X).

.

div(X,A,Y), sum(B,Y,X) <=> A~=-1 | mul(A,B,N1), sub(1,A,N2), div(N1,N2,X).

**9. Miscellaneous Simplification Constraints**
**Example:**
mul($X_1$,C,$Z_1$), mul($X_2$,C,$Z_2$) <=> sum($X_1$, $X_2$, X), sum($Z_1$, $Z_2$, Z), mul(X,C,Z).

## 6.5.4 Implementation Issues

While the theory of CHR allows more than two head constraints in constraint declarations, the implementations of CHR limit head constraints to two constraints for efficiency reasons. This limitation required us to find a way to transform rules that needed more than two head constraints.
Consider the example constraint given above in the third group of constraints:

sum(X,Y,Z), bound(X), bound(Y) ==> bound(Z).

In our implementation, we transform this constraint into a two headed constraint in two stages with the introduction of a dummy constraint.

sum(X,Y,Z), bound(X) ==> fdsum(X,Y,Z,X,0,0).
sum(X,Y,Z), bound(Y) ==> fdsum(X,Y,Z,0,Y,0).
fdsum(X,Y,Z,X,0,0), fdsum(X,Y,Z,0,Y,0) ==> bound(Z).

In addition, the semantics of CHR implementations do not match the theory exactly, therefore our implementation uses dummy constraints to avoid looping in some cases. The complete code of our implementation is provided in Appendix ?.

## 6.5.5 A Comparison with MRDSM

In MRDSM [Litwin and Vigier 86], the issue of symbolic equation solving arises when dealing with *update mappings.* The update mapping problem is defined as follows:
Let d = $f(a_1, \ldots,a_n)$ be the retrieval dynamic value obtained from the actual source attributes $a_1,\ldots,a_n$. Let d′ be the update value for d. Find the values $a'_1,\ldots,a'_n$ such that d′ = $f(a'_1, \ldots,a'_n)$.

Litwin investigates two particular problems of inversion computing, and the choice of one inversion among several, (when the mapping is many to one) combining numerical and symbolic methods. MRDSM uses the following set of arithmetic operators {+, -, *, \, **}[44] in the computation of its retrieval mappings.
Their solution for inversion computing is based on calling the generic symbolic equation solver Macsyma with
**solve({$f$(a′$_1$, …,a′$_n$) - d′ = 0}, { a′$_1$, …,a′$_n$ }).**
which calculates the values of $a'_1, \ldots,a'_n$ if $f(x_1, \ldots,x_n)$ is bijective. When Macsyma cannot solve the equation by itself, Bairstow numerical calculation method is used as a helper to Macsyma.

As seen from the above discussion, although symbolic equation solving is used in MRDSM, it is used to calculate ground values of the unknowns. Namely an *extensional solution* is sought through the combination of symbolic and numeric methods. In the case of ECOIN, however, symbolic equation solving is used to find an *intensional solution*.

[44] ** is the exponent

.

This is a crucial difference, which separates the two problems apart. In fact, the ECOIN symbolic equation solver is a superset of the MRDSM solver, for extensional answers in ECOIN can be computed after the mediated query is processed by the query executioner. Finding an *intensional solution*, however, cannot be accomplished in MRDSM.

In addition, symbolic equation solving in ECOIN is performed for multiple equations. This is also different from MRDSM, which handles one equation at a time.

Although generic symbolic equation solvers such as Macsyma, Matlab and Maple can be used to solve simultaneous equations, as in the following Maple example: `solve({3*x + 4*y = 7, 5*x + 3*y = 11}, {x,y});` the interface to their solvers have to be structured: the equations and variables to solve for has to be supplied at once. This is not easy to accomplish in a query mediation setting, where the variables to solve the equation for may not be easily identified, given the interaction between different query constructs. Furthermore, in ACLP abduction and constraint solving interfaces with each other very nicely, which is very difficult to achieve with an external symbolic solver.

Finally, our approach to the choice of inversion in many-to-one mappings is through the declaration of two separate functions -- since a many to one mapping cannot be commutative-- , which clearly identifies which mapping is to be used.

# 6.6 Illustrative Example

Consider the airfare example we have given in Chapter 4. Suppose the conversion functions shown in Figure 6.4 are defined for the *type* modifier of semantic type price .

```
cvt(commutative, price, O, type, Ctxt, "nominal", Vs, "nominal+taxes",
Vt) ←
         attr(O, tax, T),
         value(T, Ctxt, Tv),
         sum(Vs, Tv, Vt).
cvt(commutative, price, O, type, Ctxt, "nominal+taxes", Vs,
"nominal+taxes+visaFees", Vt) ←
         attr(O, provider, Pr),
         attr(Pr, visaFee, Vf),
         value(Vf, Ctxt, Vfv),
         sum(Vs, Vfv, Vt).
cvt(commutative, price, O, type, Ctxt,
"nominal+taxes+visaFees+serviceFees", Vs, "nominal+taxes+visaFees", Vt)
←
         attr(O, provider, Pr),
         attr(Pr, serviceFee, Sf),
         value(Sf, Ctxt, Sfv),
         sub(Vs, Sfv, Vt).
cvt(commutative, price, O, type, Ctxt, "final", Vs,
"nominal+taxes+visaFees+serviceFees", Vt) ←
         attr(O, provider, Pr),
         attr(Pr, paperFee, Pf),
         value(Pf, Ctxt, Pfv),
         sub(Vs, Pfv, Vt).
cvt(commutative, price, O, coverage, Ctxt, "roundtrip", Vs, "oneway",
Vt) ←
         context(O, Cs),
         modifier(price, O, coverage, Ctxt, Ms),
         value(Ms, Ctxt, “final”),
```

```
        attr(O, provider, Pr),
        attr(Pr, paperFee, Pf),
        attr(Pr, serviceFee, Sf),
        value(Pf, Ctxt, Pfv),
        value(Sf, Ctxt, Sfv),
        sum(Sfv,Pfv,SPv),
        sub(Vs, SPv, Vt1),
        div(Vt1,2,Vt2),
        sum(Vt2,SPv,Vt).
cvt(commutative, price, O, coverage, Ctxt, "roundtrip", Vs, "oneway",
Vt) ←
        context(O, Cs),
        modifier(price, O, coverage, Ctxt, Ms),
        value(Ms, Ctxt, "nominal+taxes+visaFees+serviceFees"),
        attr(O, provider, Pr),
        attr(Pr, serviceFee, Sf),
        value(Sf, Ctxt, Sfv),
        sub(Vs, Sfv, Vt1),
        div(Vt1,2,Vt2),
        sum(Vt2,Sfv,Vt).
cvt(commutative, price, O, coverage, Ctxt, "roundtrip", Vs, "oneway",
Vt) ←
        div(Vs,2,Vt).
```

Figure 6.4 Conversion Functions for Modifier *type*

These functions correspond a network shown in Figure 6.5. The arcs roughly show which arithmetic predicates are used to do the conversion from one modifier value to another. Note that the coverage and type modifiers of price are not completely orthogonal, and order dependent. Namely, the type modifier has precedence over the coverage modifier. ECOIN system relies on the order of specification of modifiers to deal with this case. The other approach would be to combine the type and coverage modifiers into a single modifier, but this would almost double the number of conversion functions needed to do conversion.

Consider now the following user query we have introduced in Chapter 4:

Q2: SELECT Airline, Price FROM Yahoo
WHERE DepDate = "01/06/03" and ArrDate= "01/08/03"
and DepCity= "Boston" and ArrCity= "Istanbul";

This query is converted to the following well formed query again introduced in Chapter 4:

WFQ2: answer(VAirline,VPrice) ←
yahoo′(_,Airline′, Price′, _, DDate′, ADate′ , DCity′, _, ACity′),
value(Price′, c_user, VPrice)[45],
value(Airline′,c_user, VAirline),
value(DDate′,c_user, "01/06/03"),
value(ADate′,c_user, "01/08/03")
value(DCity, c_user, "Boston")

[45] Read as "the value of semantic object Price′ in context c_user is Price".

value(ACity, c_user, “Istanbul”).

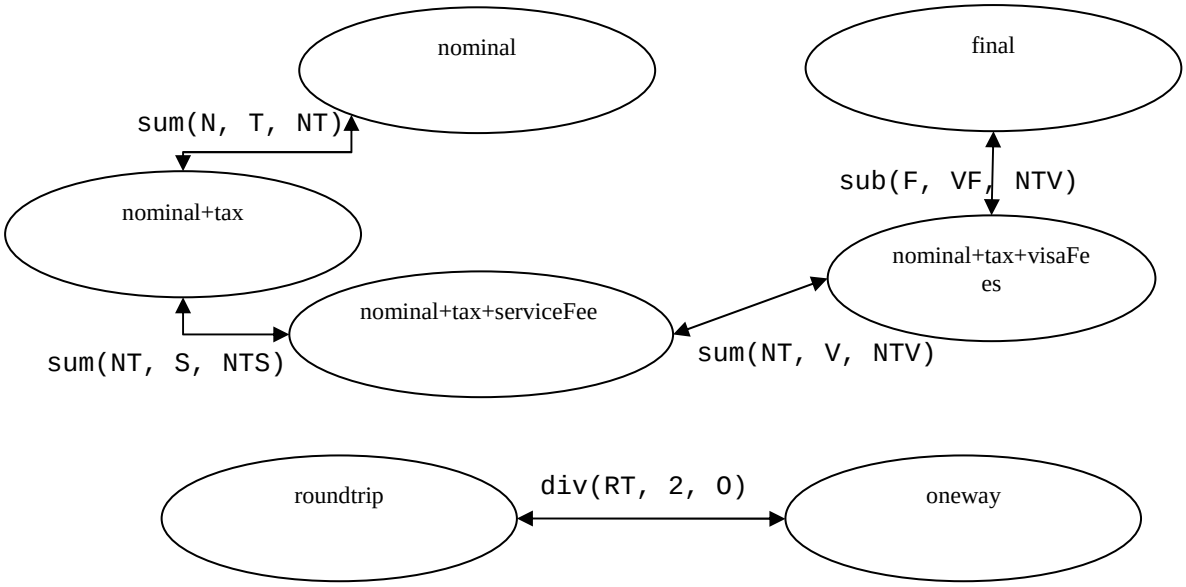



Figure 6.5 A conversion function network for modifier type and coverage

After this query goes through the abduction reasoning as in [Goh 97], the following constraint predicates are posted to the constraint store:

{{ answer(VAirline,VPrice),
yahoo(I,VAirline, Price, T, “06/01/03”, “08/01/03”, $Airport_1$, “Great Britain”, $Airport_2$),
visafees(“Transit, UK, Turkey”, VF),
cityAirport(“Boston”, $Airport_1$),
cityAirport(“Istanbul”, $Airport_2$),
currencyconvert(“GBP”,”USD”, ExchangeRate, “05/01/03”),
sum(Price, T, PT),
mul(ExchangeRate,VF,VFA),
sum(PT,VFA,PTV),
sub(PTVS,5,PTV),
sub(Final, 20, PTVS),
sum(20,5,SP),
sub(Final, SP, FSP),
div(RT,2,FSP),
sum(RT,SP,VPrice).
}{ answer(VAirline,VPrice),
yahoo(I,VAirline, Price, T, “06/01/03”, “08/01/03”, $Airport_1$, Cxn, $Airport_2$),
Cxn <> “Great Britain”,
cityAirport(“Boston”, $Airport_1$),
cityAirport(“Istanbul”, $Airport_2$),
sum(Price, T, PT),
sum(PT,0,PTV),

.

```
   sub(PTVS,5,PTV),
   sub(Final, 20, PTVS),
   sum(20,5,SP),
   sub(Final, SP, FWOP),
   div(RT,2,FWOP),
   sum(RT,SP,VPrice).
}
}
```

Note that because there are two subsets of abducibles, the mediated query will be a union of these. Let's consider the first set of abducibles and the CHR rules together:

```
   sum(Price, T, PT),
   mul(ExchangeRate,VF,VFA),
   sum(PT,VFA,PTV),
   sub(PTVS,5,PTV),
   sub(Final, 20, PTVS),
   sum(20,5,SP),
   sub(Final, SP, FSP),
   div(RT,2,FSP),
   sum(RT,SP,VPrice).
```

In processing the above set of constraints, the following CHR rules are used:

**Rule 1: If a variable is ground it is also bound**

sub(Final,20,PTVS) ==> ground(20) | **bound(20).**
sub(PTVS,5,PTV) ==> ground(5) | **bound(5).**
div(RT,2,FWOP) ==> ground(2) | **bound(2)**.

**Rule 2: If a variable is functionally determined by ground values, it is bound and its value can be calculated**

sum(20,5,SP), bound(20), bound(5) <=> SP is 25.

**Rule 3 : If a result variable is functionally determined by bound values, it is also bound**

sum(Price, T, PT), bound(Price), bound(T) ==> bound(PT).
*T and Price are from the yahoo relation; therefore are intensionally bound*.
mul(ExchangeRate,VF,VFA), bound(ExchangeRate), bound(VF) ==> bound(VFA).
*ExchangeRate and VF are from the currencyconvert, and visaFees relations; therefore are intensionally bound.*
sum(PT,VFA,PTV), bound(PT), bound(VFA) ==> bound(PTV).

**Rule 6: If a result variable in a predicate is bound simplify that predicate with its inverse**

sub(PTVS,5,PTV), bound(PTV) <=> sum(PTV,5,PTVS), bound(PTVS).
sub(Final, 20, PTVS), bound(PTVS) <=> sum(PTVS,20,Final), bound(Final).

**Rule 3 : If a result variable is functionally determined by bound values, it is also bound**

sub(Final, 25, FSP), bound(Final), bound(25) ==> bound(FSP).

.

**Rule 6: If a result variable in a predicate is bound simplify that predicate with its inverse**
div(RT,2,FSP), bound(FSP) <=> mul(FSP,2,RT), bound(RT).

**Rule 3: If a result variable is functionally determined by bound values, it is also bound**
sum(RT,25,VPrice), bound(RT), bound(25) ==> bound(VPrice).

At this stage since all variables are bound we can stop. The following set of arithmetic predicates

sum(Price, T, PT),
mul(ExchangeRate,VF,VFA),
sum(PT,VFA,PTV),
sum(PTV,5,PTVS),
sub(PTVS, 20, Final),
sub(Final, 25, FSP),
mul(FSP,2,RT),
sum(RT,25,VPrice).

can be used to construct:
VPrice = (Price +Tax + ExchangeRate* VisaFee + 5 + 20 -25) * 2 + 25
Further simplifications can be done by using more simplification constraints. For example we can have the following propagation
sum(PTV,5,PTVS), sum(PTVS, 20, Final) ==> sum(PTV,25,Final).
by using:
**Rule 9. Miscellaneous Simplification Constraints**
**sum(X,Y,Z), sum(Z,A,B) ==> ground(Y), ground(A), nonground(X), nonground(B) | C is Y + A, sum(X, C, B).**
The choice of simplification vs. propagation is important here, because before we can use a simplification we have to make sure that Z is not referred by any other constraint predicate. This would require using another constraint such as notreferred(X) to keep track of variable referrals. In that case the above propagation rule could be written as a simplification:
**sum(X,Y,Z), sum(Z,A,B), notreferred(Z) <=> ground(Y), ground(A), nonground(X), nonground(B) | C is Y + A, sum(X, C, B).**
Furthermore we can have the following simplification:
sum(PTV,25,Final), sub(Final, 25, FSP) <=> FSP=PTV, sum(PTV,25,Final).
by using
**Rule 5: Arithmetic integrity constraints**
sum(X,A,Y), sub(Y, A, Z) <=> X=Z, sum(X,A,Y) or
sum(X,A,Y), sub(Y, A, Z) <=> X=Z, sub(Y,A,X)
Finally we end up with the following set of abducibles:
{
{ answer(VAirline,VPrice),
yahoo(I,VAirline, Price, T, “06/01/03”, “08/01/03”, $Airport_1$, “Great Britain”, $Airport_2$),
visafees(“Transit, UK, Turkey”, VF),

.

cityAirport(“Boston”, $Airport_1$),
cityAirport(“Istanbul”, $Airport_2$),
currencyconvert(“GBP”,”USD”, ExchangeRate, “05/01/03”),
sum(Price, T, PT),
mul(ExchangeRate,VF,VFA),
sum(PT,VFA,PTV),
mul(PTV,2,RT),
sum(RT,25,VPrice).

}
{ answer(VAirline,VPrice),
yahoo(I,VAirline, Price, T, “06/01/03”, “08/01/03”, $Airport_1$, Cxn, $Airport_2$),
Cxn <> “Great Britain”,
cityAirport(“Boston”, $Airport_1$),
cityAirport(“Istanbul”, $Airport_2$),
sum(Price, T, PT),
mul(PT,2,RT),
sum(RT,25,VPrice).
}
}
which translates to the mediated query we have shown in Chapter 4:

```
MQ2:  SELECT Airline, 2* (Price+Tax+ VisaFee*exchangeRate) + 25
      FROM yahoo, visafees, currencyconvert,
      (select Airport from cityAirport where city= “Boston”) depCode,
      (select Airport from cityAirport where city= “Istanbul”) arrCode,
      WHERE DepDate = “06/01/03” and ArrDate= “08/01/03” and
      DepCity= depCode.Airport and ArrCity= arrCode.Airport
      and CxnCountry= “Great Britain” and fromCur=”GBP”
      and toCur=”USD” and date= “5/01/03”;
      UNION
      SELECT Airline, 2* (Price+Tax) +25
      FROM yahoo, visafees,
      (select Airport from cityAirport where city= “Boston”) depCode,
      (select Airport from cityAirport where city= “Istanbul”) arrCode,
      WHERE DepDate = “06/01/03” and ArrDate=”08/01/03” and
      DepCity= depCode.Airport and ArrCity= arrCode.Airport
      and CxnCountry <> “Great Britain”;
```

.

# Chapter 7

# Ontology and Context Merging in ECOIN

With the ECOIN framework users can create applications that integrate disparate data sources. Many of these applications are domain specific and can offer more value if they can be used together. Consider for example using the ECOIN airfare application we have presented in Chapter 4, together with a car rental application to plan an integrated travel schedule. Often times, however, these applications are developed by disparate users with diverse backgrounds, therefore cannot be immediately integrated.

Integrating disparate ECOIN applications is challenging because it involves integrating disparate ontologies as well as context frames defined by them. It requires an understanding of the semantics of application ontologies and their context frames. Furthermore, extensions to the ontologies and context frames may be necessary to address the semantic conflicts emerging in the merged applications. For example, while the currency may not be part of the ontology and/or context frame in two applications focusing solely on US or European sources, it has to be made explicit when the applications are merged. Thus the context frame of the new ontology has to have a currency modifier.

We adopt a two at a time *virtual merging* approach to integrate disparate ECOIN applications. The merger application does not physically contain the applications it merges, but only axioms that are needed to align and extend them. Unlike ontology merging approaches in the literature, our approach is context driven and primarily requires context frames of ontologies to be merged.

We start this chapter by providing a brief literature review on integrating ontologies. Then, we explain the basics of our merging process with the example scenario of creating a travel application from the airfare application of Chapter 4 and newly introduced car rental example. Finally, we provide a formal framework for the virtual merging of ECOIN applications. This framework is used in [Kaleem 03] as the foundation of a graphical tool that facilitates integrating disparate ECOIN explanations.

## 7.1 Literature Review on Integrating Ontologies

The origins of ontology integration can be traced back to schema integration, which has been studied extensively since eighties [Batini and Lenzerini 86]. Schema integration

research produced some guidelines to be used in integrating disparate schemas and semi-automatic tools, but the process could not be fully automated because schema semantics could not be made explicit without human intervention.

Ontology integration has to deal with both syntactic and semantic heterogeneities. Syntactically, ontologies may be expressed using different languages (e.g. KL-ONE vs. KIF) that may have different level of expressiveness (e.g. one supports default values the other does not). Ontolingua project [Gruber 93] aims to overcome this problem by providing an ontology language that can be translated to a variety of other ontology languages through the use of special purpose translators. It also provides a centralized repository to encourage reuse of ontologies developed in a variety of languages.

Semantic differences such as the ones shown in Figure 7.1 are more difficult to reconcile because they require human intervention to understand and reconcile the meaning of ontological terms and relationships.

- Terminological Differences
  - Different names for the same concept
  - Related but different concepts
  - More specialized or general versions of the same concept
  - Attributes *vs*. functions *vs.* predicates representation
- Simple Structural Differences
  - Two ontologies are similar yet disjoint
  - One ontology is a subset of the other
  - One ontology is a reorganization of the other
- Complex Structural Differences
  E.g., having action predicates *vs.* reified events
- Fundamentally different representations
  E.g., Bayesian probabilistic *vs.* truth-logic

Figure 7.1 List of differences between ontologies

(Adopted from Reed and Lenat 02)

Efforts such as the Standard Upper Ontology (SUO) [Niles and Pease 01] and Cyc Upper Ontology [Reed and Lenat 02] aim to reduce this need and provide general ontologies that can be used as the foundation of more specific ontologies. In these efforts, mappings that translate concepts of one ontology into the standard upper ontology are defined. The Carnot project for instance maps domain specific schemas to the Cyc knowledge base through the use of articulation axioms.

These articulation axioms may relate synonymous concepts with each other as shown below:

(synonymousExternalConcept Waikohu-CountyNewZealand FIPS10-4Information1995 "NZ86")

where Waikohu-CountyNewZealand is the Cyc term synonymous with “NZ86” in source FIPS10-4Information1995.

Or they may specify an overlapping relation as in the following example:

.

(overlappingExternalConcept InferiorMesentericVein MeSH-Information1997
"Mesenteric Veins | A7.231.908.670.385")

Approaches that integrate ontologies by defining mappings (e.g. articulation axioms in Carnot) between them are known as *ontology alignment* approaches. On the other hand approaches that aim to produce a new ontology out of a set of ontologies is known as *ontology merging*. We consider integrating ontologies in ECOIN as a hybrid of these approaches: like ontology alignment approaches we use articulation axioms to align ontologies, and like ontology merging approaches we produce a new (*virtual*) ontology out of two ontologies.

The state of the art in ontology merging today is dominated by semi-automatic tools that can analyze ontologies, and guide the user during merging by making suggestions. Three well known such tools are Prompt [Noy and Musen 00], Ontomorph [MacGregor et al. 99] and Chiemera [McGuinness et al. 00].

In all of these approaches, the first step is the *syntactic match* phase in which ontological terms referring to similar objects are identified based on a linguistic similarity measurement. In the simplest case synonyms from a thesaurus can be used. In more sophisticated approaches, a lexical reference system like Wordnet [Miller 95] can be used to identify similar terms through the use of richer semantics that involves relationships linking different synonyms sets.

In Ontomorph [Chalupsky 00], which is based on the PowerLoom knowledge representation system, the user is offered a number of transformative operators to apply to the initial list of matches from the syntactic match phase. A human expert has to do the rest of merging manually. Chimaera [McGuinness et al. 2000] is like Ontomorph but considers subclass-superclass relations when making suggestions. PROMPT, previously known as SMART, is built on top of an ontology editor tool Protégé 2000. Based on a linguistic similarity among concept names it suggests actions, which may be applied by the user. It also allows users to define new actions by using the Protégé 2000 tool.

These tools are useful in cutting some amount work during ontology merging, but because the semantics of different ontologies cannot automatically be made explicit, the user still has the burden of understanding each ontology before doing the merging.

## 7.2 Example Merging Scenario

In Chapter 4, we described the airfare application that helps users find the cheapest airfare across multiple airfare providers such as Orbitz, Expedia, Yahoo, Qixo. In this section, we introduce a car rental application that is used to find the cheapest car rental prices across multiple car rental providers. Then we explain the process of merging the airfare and car rental applications.

### 7.2.1 Car Rental Scenario

After finding the cheapest airfare, our thrifty friend wants to rent a car from the airport. Luckily, there is a car rental ECOIN application developed to find the best rental prices from a number of online providers. The details of this scenario are illustrated in Figure 7.2. As seen in the figure the context of the user and the car rental data source conflict in several ways. For example, in the user context the price means the final price (including taxes and fees), whereas in the data source price is nominal.

## Context of Expediacar

Price means *nominal price*

Rate period is daily, weekend, weekly, or monthly

Several fees are charged
e.g. Vehicle license fee... 2.75/DAY
Airport concession recovery fee ... 10 percent
GARS 5/Day
04/01/03 - 10/18/03 peak season surcharge... 3.00/DAY

Pickup and Dropoff locations are expressed as
*three letter airport codes*

Currency = US Dollars  Date= American

## Context of User

Price means *final price*
(including taxes, and fees)

Rate period is *duration of rental*

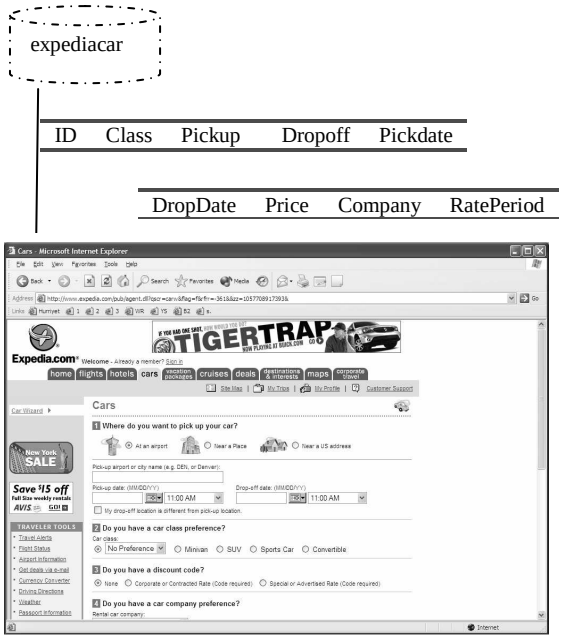
Pickup and Dropoff locations are expressed as *city names*

Currency = US Dollars

Date= American

Query

SELECT Price FROM expediacar
WHERE Class= “Economy” and PickDate = “01/08/03” and DropDate= “03/08/03” and Pickup=”Boston” and DropOff= “Boston”;



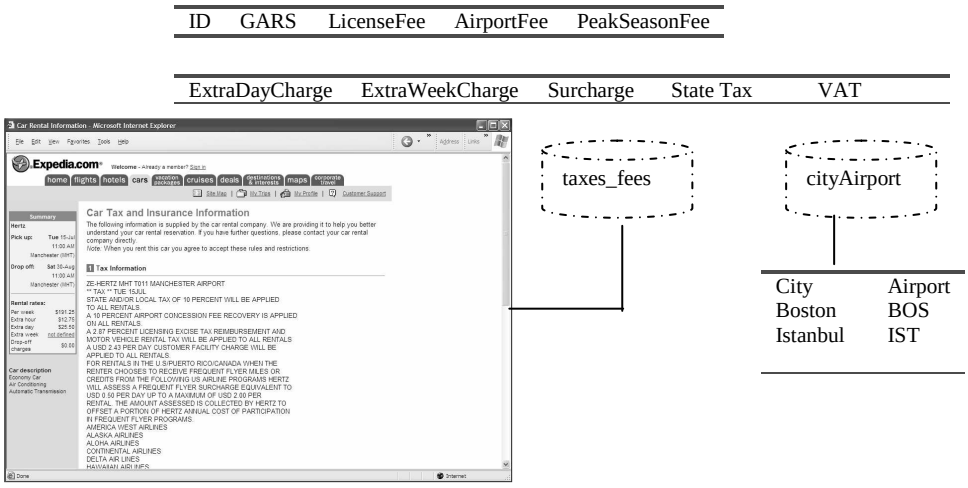




Figure 7.2 Car Rental Example Scenario

.

Rate period in expediacar can be daily, weekend, weekly, or monthly and is revealed with the results, but the user expects that the returned rate will be the total rate for the whole duration.

Our user formulates the following query to find rental prices:

```
Q1:   SELECT Company, Price FROM expediacar
      WHERE Class = “Economy” and PickDate = “06/02/03” and
      DropDate=  “08/01/03”  and Pickup=”Istanbul” and DropOff= “Istanbul”;
```

This query requests from expediacar the price and provider company of economy cars that can be picked up and dropped off in Istanbul on June 2nd 2003 and August 1st 2003 respectively. This naïve query, when submitted to ECOIN, would be rewritten into the following mediated query:

```
MQ1:  SELECT Company, (Price * 2 + ExtraDayCharge) * (1 + VAT)
      FROM expediacar, taxes_fees
      (select Airport from cityAirport where city= “Istanbul”) cityCode,
      WHERE Class= “Economy” and PickDate = “06/01/03” and
      DropDate=”08/01/03” and   PickUp= cityCode.Airport and
      DropOff= cityCode.Airport and expediacar.ID=taxes_fees.ID;
```

and the system would return a result set like:

| **Company** | **Price** |
|---|---|
| Hertz | 2328 |
| National | 2768 |
| … | … |

As seen in MQ1, the mediation engine calculates the total price by multiplying the monthly price returned by the source by 2 (covering June and July) and adding an extra day charge (one day from August). Furthermore, the price is adjusted to include the taxes by dynamically obtaining the value added tax (VAT) for Istanbul.

While car rental and airfare applications can be queried individually using ECOIN, the user cannot issue a query that refers to both applications without merging them. The following query, for example, cannot be issued unless these applications are merged:

```
      SELECT yahoo.Airline, expediacar.Company, yahoo.Price + expediacar.Price as
total
      FROM yahoo, expediacar
      WHERE DepDate = “01/06/03” and ArrDate=  “01/08/03”
      and  DepCity= “Boston” and ArrCity= “Istanbul”;
      expediacar.Pickup="Istanbul"  and  expediacar.Dropoff="Istanbul"  and
      expediacar.PickDate="01/06/03" and expediacar.DropDate="08/01/03";
```

The query above asks for the airline and rental companies with the total price of airfare (from Boston to Istanbul between June 1st and August 1st in 2003) and car rental (pick up and drop off in Istanbul between June 1st and August 1st in 2003). Next we explain the merging process by using the airfare and car rental applications.

.

### 7.2.2 Merging Airfare and Car Rental

In order to use the airfare and car rental applications together, we need to create a new application that merges the two. This new application, named the Travel Application in Figure 7.3, defines mappings between airfare and car rental ontologies, context frames and conversion functions.

The Travel Application is not only a store for defining mappings between airfare and car rental applications, but also a normal application that can have its own ontological extensions, new context frames and conversion functions.

For the user it is just any other application, as the user is not aware of the underlying applications as shown in Figure 7.4. The mediation engine on the other hand uses underlying applications when answering a user query since the travel application does not physically contain either application.

As shown in Figure 7.5, merging is done two at a time. This choice enables users to gradually detect and resolve conflicts as well as simplifying the reasoning algorithm used in merging. Arbitrary number of applications, however, can be merged since merger applications are no different than other applications and can participate in new mergings.

Merging in ECOIN is driven by the need to merge context frames and the conversion functions that apply to modifiers. Ontology merging is needed because context frames and conversion functions are defined by ontologies. In the extreme case of two applications having no contextual conflicts, there is no need to merge ontologies to answer queries referring to these applications. This can be understood better, if we note first that the abductive query answering in ECOIN works (in procedural terms) by

- recursively finding the modifiers of a semantic object
- applying conversion functions for each modifier when needed;

and second the queries are expressed not using the shared ontology, but by referring to source schemas similar to those in loosely coupled approaches.

For this reason, in the extreme case of two applications sharing identical contexts, there is no need to reconcile ontologies even when they exhibit differences stated in Figure 7.1. In practice, however, two applications are likely to have contextual conflicts, and affected parts of ontologies have to be aligned. The context oriented merging provides an important advantage over ontology oriented merging (as in tightly-coupled systems), for it minimizes the amount of conflict resolution between disparate ontologies.

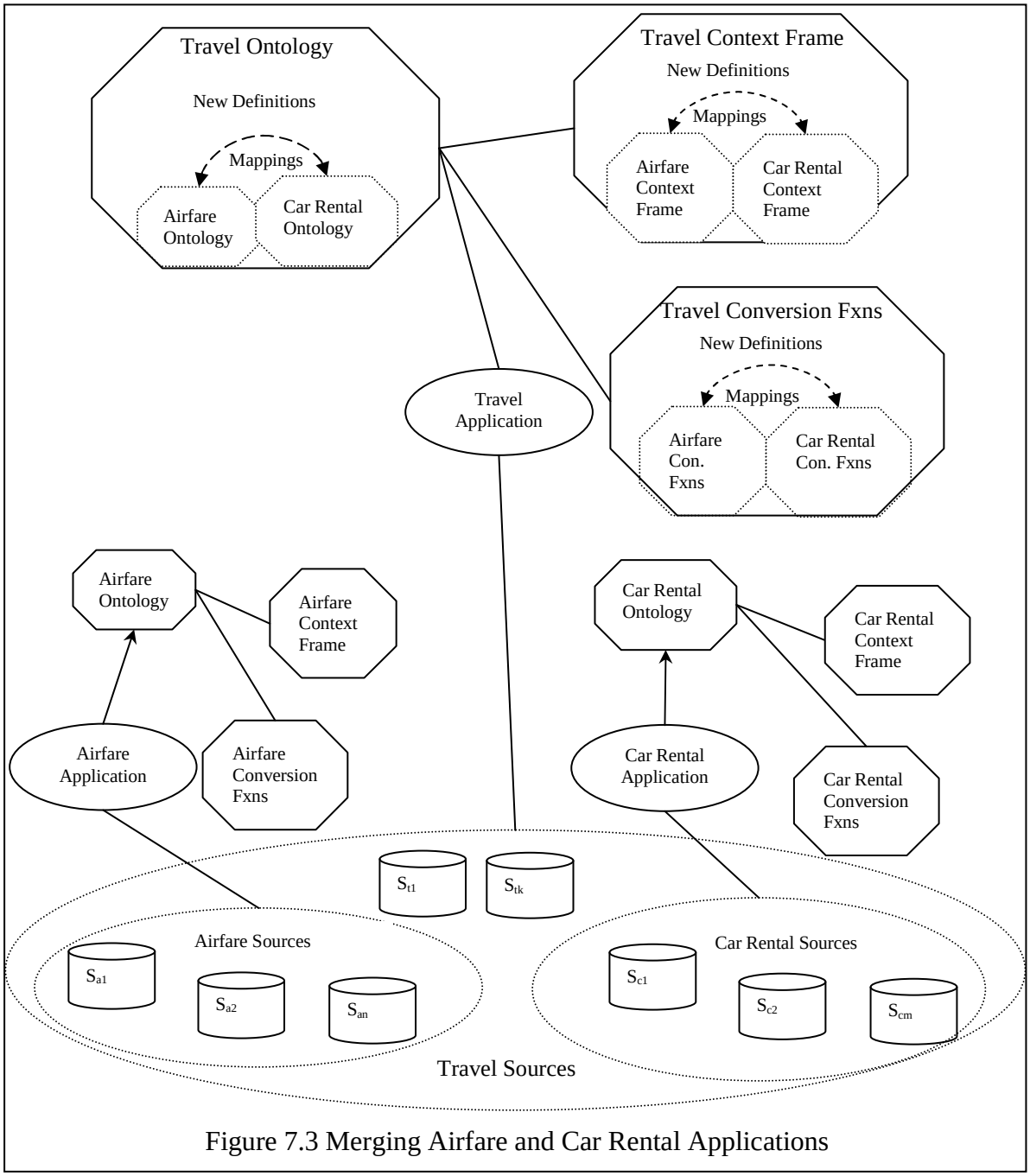



Figure 7.3 Merging Airfare and Car Rental Applications

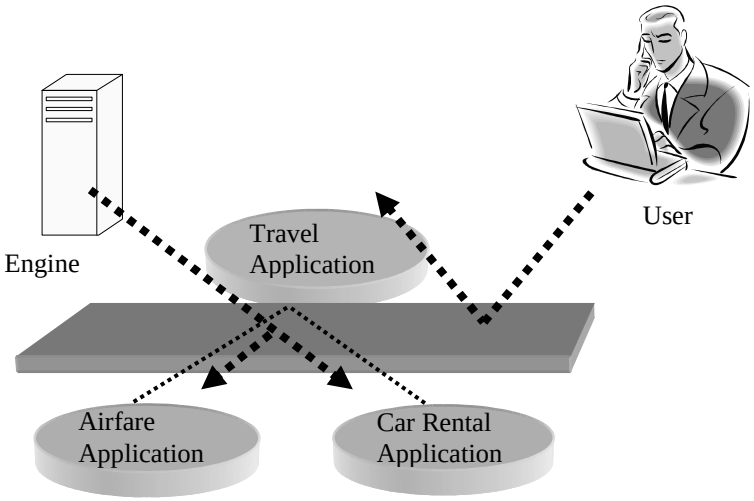



Figure 7.4 User and engine visibility in merging

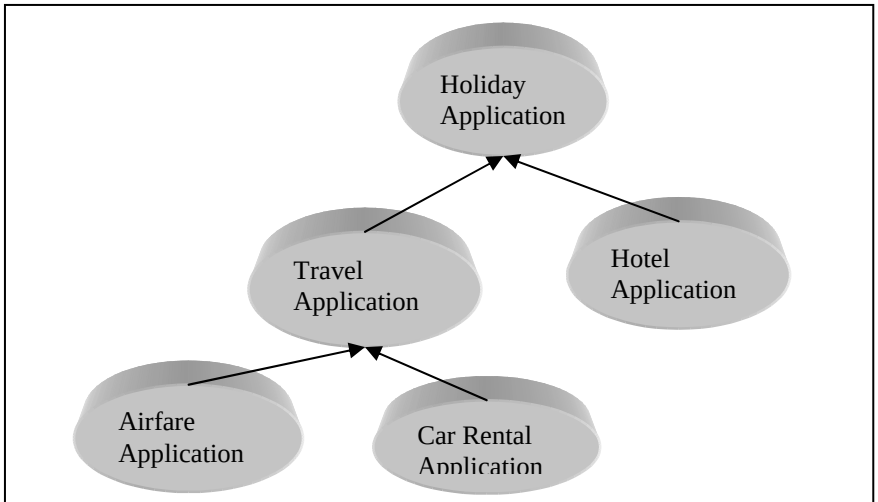



Figure 7.5 Two at a time merging

# 7.3 Knowledge Representation for Merging

The ECOIN Merging Framework (ECOINM) extends the ECOIN framework to allow the merging of multiple ECOIN or ECOINM applications. To the users the resulting ECOINM application is indistinguishable from any other ECOIN application. From the system point of view, the resulting application is a virtual one that has links to the underlying applications, but does not physically contain them. Instead it includes axioms that relate the context frames of those applications' ontologies. Below we provide the details of knowledge representation for merging.

.

## 7.3.1 Notation and Assumptions

The application that merges two other applications is called the *merger application* of those applications. Applications that are merged are called the *merged applications*. A merger application has *physical and logical views*. Physical view of the merger application includes declarations that are physically contained in the merger application, whereas the logical view includes all declarations that can be logically deduced. Notation for the physical view will be the same notation we used in the ECOIN framework. Notation for the logical view will assume the form of $X@A_i$ where X is a term that belongs to physical notation and $A_i$ denotes the application id. For example, while T will denote the semantic types in the physical view of the subject application, $T@A_i$ will correspond to semantic types in the logical view of application $A_i$. Finally, we assume that all names (semantic type names, attribute names, etc.) used in the applications are unique or made unique with the use of an appropriate scheme (e.g. URIs).

## 7.3.2 Declarations

**Definition** (Merger Declarations)
Let A be the application that merges applications $A_1$ to $A_n$[46].

- A *merging relationship* that specifies the merger and the merged applications

  $$\text{merges}(A, [A_1,A_2])$$

  This is read as: Application A is the *merger* root of applications $A_1$ and $A_2$.
- An *isomodifiertype relationship* that specifies the semantic type mappings between the merger and the merged applications

  $$\text{isomodifiertype}(A,A_i,\tau,\tau_{ij})$$

  This is read as: Semantic type $\tau$ in application A and semantic type $\tau_{ij}$ in application $A_i$ has compatible modifiers.
- An *isomodifier relationship* that specifies the modifier name mappings between the merger and the merged applications

  $$\text{isomodifier}(A, A_i, m, m_{ij})$$

  This is read as: Modifier m ($m \in M(\tau)$ ) in application A and modifier $m_{ij}$ in application $A_i$ are equivalent modifiers)
- An *isocontext relationship* that specifies the context identifier mappings between the merger and the merged applications

  $$\text{isocontext}(A,A_i,c,c_{ij})$$

  This is read as: Context c in application A and context $c_{ij}$ in application $A_i$ are equivalent contexts
- An *isoattribute relationship* that specifies the attribute name mappings between the merger and the merged applications

  $$\text{isoattribute}(A,A_i,a, a_{ij})$$

  This is read as:Attribute a ($a \in A(\tau)$) in application A and attribute $a_{ij}$ in application $A_i$ are equivalent attributes

Note that the above mappings are always specified between the merger and the merged applications, never between merged applications directly.

[46] For simplicity reasons we are going to take n=2 in the rest of the discussion.

.

**Definition** (Default Transitions)
isomodifiertype(A,$A_i$, $\tau'$, $\tau$') ← $\tau$' ∈ T@$A_i$, not isomodifiertype(A,$A_i$, $\tau$, $\tau$'), $\tau \neq \tau$'.
isocontext(A,$A_i$,c′,c′) ← c' ∈ C@$A_i$, not isocontext(A,$A_i$, c, c'), c ≠ c′.
isomodifier(A, $A_i$, m′, m′ ) ← m' ∈ M($\tau$)@$A_i$,not isomodifier(A, $A_i$, m, m′), m ≠ m′.

*Informally*, this means that any semantic type (or context, modifier) in the merged applications that is not explicitly mapped to the merger application by default exists in the merger application with its own name.

**Definition** (Mappings)
Mappings are polymorphically defined as follows:

- Semantic type mapping
  map(A,$A_i$, $\tau$, $\tau_{ik}$) ← isomodifiertype(A,$A_i$,$\tau$,$\tau_{ij}$)
- Modifier mappings
  map(A,$A_i$, m, $m_{ij}$ ) ← isomodifier(A, $A_i$, m, $m_{ij}$)
- Context mappings
  map(A,$A_i$, c, $c_{ik}$) ← isocontext(A,$A_i$,c,$c_{ij}$)
- Attribute mappings
  map(A,$A_i$, c, $c_{ik}$) ← isoattribute(A,$A_i$,c,$c_{ij}$)
- Object mappings
  map(A,$A_i$, t, $t_{ik}$) ← t = skolem($\tau$, $t_j$, c, j, r($t_1$,…, $t_n$)),
  $t_{ik}$ = skolem($\tau_{ik}$, $t_j$, $c_{il}$, j, r($t_1$,…, $t_n$)),
  map(A,$A_i$, $\tau$, $\tau_{ik}$), map(A,$A_i$, c, $c_{il}$).

## 7.3.3 Context

This section contains context related definitions. These definitions are similar to those given in Chapter 5, except that these definitions are from the logical view, i.e. considers both the physical and virtual declarations in computationally describing context related concepts.

**Definition** (*Logical Context Frame of a Semantic Type*)
Let $\tau$ be a semantic type in application A that merges applications $A_1$ and $A_2$. The logical context frame of $\tau$, M($\tau$)@A is defined as a set as follows:
m ∈ M($\tau$)@A ← m ∈ M($\tau$).
m ∈ M($\tau$) @A← merges(A, MSet), $A_i$ ∈ MSet,
map(A,$A_i$, $\tau$, $\tau_{ik}$), map(A, $A_i$, m, m′),
m′∈ M($\tau_{ik}$)@$A_i$.

**Definition** (*Logical Extensional Context of a Semantic Object*)
Let t be a semantic object of type $\tau$ in application A, the logical extensional context c of object t, $C_E$(t,c)@A, is defined as a set as follows:
{m, $t_v$}∈ $C_E$ (t,c)@A ←m ∈ M($\tau$), {m, $t_v$} ∈ $C_E$(t,c).

{m,$t_v$} ∈ $C_E$ (t,c)@A ←m ∈ M($\tau$), merges(A, MSet), $A_i$ ∈ MSet,
map(A, $A_i$, t, $t_i$), map(A, $A_i$, c, $c_i$), map(A, $A_i$, m, $m_{ij}$),
{$m_{ij}$, $t_v$}∈ $C_E$ ($t_i$,$c_i$)@$A_i$, {m, $t_v$} ∉ $C_E$(t,c).

**Definition** (*Logical Intensional Context of a Semantic Object*)
Let t be a semantic object of type $\tau$ in application A, $t'$ be a semantic object, the logical intensional context c of object t, $C_I(t, c)@A$, is defined as a set as follows:

$x \in C_I(t,c)@A \leftarrow x \in C_I(t,c)$.

$x \in C_I(t,c)@A \leftarrow$ merges(A, MSet), $A_i \in$ MSet,
$x = (\text{modifier}(\tau, t, m, c, t') \leftarrow L_1, \ldots, L_n)$,
$x_i = (\text{modifier}(\tau', t_i, m_i, c_i, t'_i) \leftarrow L'_1, \ldots, L'_n)$,
$\text{map}(A, A_i, t, t_i)$, $\text{map}(A, A_i, t', t'_i)$, $\text{map}(A, A_i, \tau, \tau')$
$\text{map}(A, A_i, c, c_i)$, $\text{map}(A, A_i, m, m_i)$,
$x_i \in C_I(t_i,c_i)@A_i$, $x \notin C_I(t,c)$.

**Definition** (*Logical Context Referred by Identifier*)
Logical extensional and intensional context referred by an identifier c, $C_E(c)@A$, $C_I(c)@A$ is defined as follows:

$\{\tau,m,v\} \in C_E(c)@A \leftarrow \{\tau,m,v\} \in C_E(c)$.

$\{\tau,m,v\} \in C_E(c)@A \leftarrow$ merges(A, MSet), $A_i \in$ MSet,
$\text{map}(A, A_i, \tau, \tau_i)$, $\text{map}(A, A_i, m, m_i)$,
$\{\tau_i,m_i,v\} \in C_E(c)@A_i$, $\{\tau,m,v\} \notin C_E(c)$.

$C_I(c)@A \supseteq C_I(c)$.

$C_I(c)@A \supseteq C_I(c) \leftarrow$ merges(A, MSet), $A_i \in$ MSet,
$x = (\text{modifier}(\tau, t, m, c, t') \leftarrow L_1, \ldots, L_n)$,
$x_i = (\text{modifier}(\tau_i, t_i, m_i, c_i, t'_i) \leftarrow L'_1, \ldots, L'_n)$,
$\text{map}(A, A_i, t, t_i)$, $\text{map}(A, A_i, t', t'_i)$, $\text{map}(A, A_i, \tau, \tau_i)$,
$\text{map}(A, A_i, c, c_i)$, $\text{map}(A, A_i, m, m_i)$,
$x_i \in C_I(c_i)@A_i$, $x \notin C_I(c)$.

**Definition** (*Logical Context Frame of an Ontology*)
Logical context frame of an ontology O, is a set defined as follows:

$\langle \tau, C(\tau)@A \rangle \in C(O)@A \leftarrow C(\tau)@A \neq \emptyset, \tau \in T@A$.

## 7.3.4 Ontology

**Definition** (*Semantic Types in ECOINM Framework*)
Let A be the application that merges applications $A_1$ and $A_2$. Semantic types of A, T@A is defined as follows:

$\tau \in T@A \leftarrow \tau \in T$ .

$\tau \in T@A \leftarrow$ merges(A, MSet), $A_i \in$ MSet, $\tau_{ij} \in T@A_i$, $\text{map}(A, A_i, \tau, \tau_{ij})$.

**Definition (***Attributes in ECOINM Framework*)
Let $\tau$ be a semantic type in T@A, the logical attributes of $\tau$, $A(\tau)@A$, is defined as follows:

$a \in A(\tau)@A \leftarrow a \in A(\tau)$.

$a \in A(\tau)@A \leftarrow$ merges(A, MSet), $A_i \in$ MSet, $\tau_{ij} \in T@A_i$,
$\text{map}(A, A_i, \tau, \tau_{ij})$, $\text{map}(A, A_i, a, a_{ik})$, $a_{ik} \in A(\tau_{ij})@A_i$.

.

**Definition (***Is-a relationships in ECOINM Framework*)
Let $\tau$ be a semantic type in T@A or a context identifier in C@A, the logical is-a relationships of application A, H@A, is defined as follows:
$x \in H\ @A \leftarrow x \in H$.
$x \in H\ @A \leftarrow$ merges(A, MSet), $A_i \in$ MSet, $x = is\_a(\tau,\tau')$, $x_i = is\_a(\tau_{ij},\tau'_{ik})$,
map(A, $A_i$, $\tau,\tau_{ij}$), map(A, $A_i$, $\tau',\tau'_{ik}$), $x_i \in H\ @A_i$ .

**Definition** (*Logical Ontology in ECOINM framework*)
The logical ontology O@A in ECOINM framework is T @A $\cup$ C@A $\cup$ H@A $\cup$ A@A $\cup$ M@A , where

- A is the set of declarations of $a \in A(\tau)@A$, $\tau \in$ T@A
- M is the set of declarations of $m \in M(\tau)@A$, $\tau \in$ T@A

## 7.3.5 Rest of the Concepts

### Conversion Functions

**Definition** (*Logical Conversion Functions of an Ontology*)
Logical conversion functions of an ontology O, CF(O)@A, is defined as follows:
$f$ ($t$, $t_v$, m, c, $m_{vs}$, $m_{vt}$) $\in$ CF(O)@A$\leftarrow$ $f$ ($t$, $t_v$, m, c, $m_{vs}$, $m_{vt}$) $\in$CF(O) .
$f$ ($t$, $t_v$, m, c, $m_{vs}$, $m_{vt}$) $\in$ CF (O)@A$\leftarrow$ merges(A, MSet), $A_i \in$ MSet,
map(A, $A_i$, $t$, $t_{ij}$), map(A, $A_i$, m, $m_{ik}$), map(A, $A_i$, c, $c'_{il}$),
$f$ ($t_{ij}$, $t_v$, $m_{ik}$, $c_{il}$, $m_{vs}$, $m_{vt}$) $\in$CF(O)@$A_i$ .

### Sources

**Definition** (*Logical Source Set of an Ontology*)
Logical source set of an ontology O, S(O)@A, is defined as follows
$s \in S(O)@A \leftarrow s \in S(O)$.
$s \in S(O)@A \leftarrow$ merges(A, MSet), $A_i \in$ MSet, $s \in S(O)@A_i$.

### Elevations

**Definition** (*Logical Elevation Set of an Ontology*)
Logical elevation set of an ontology O, S(O)@A, is defined as follows
$e \in E(O)@A \leftarrow e \in E(O)$.
$e \in E(O)@A \leftarrow$ merges(A, MSet), $A_i \in$ MSet, $e \in E(O)@A_i$.

### Constraints

**Definition** (*Logical Constraint Set of an Ontology*)
Logical source set of an ontology O, S(O)@A, is defined as follows
$ic \in IC(O)@A \leftarrow ic \in IC(O)$.

.

$ic \in IC(O)@A \leftarrow merges(A, MSet), A_i \in MSet, ic \in IC(O)@A_i$.

### ECOINM Framework

**Definition** An ECOINM framework is $S@A \cup O@A \cup E@A \cup C@A \cup CF@A \cup CS@A$ .

**Definition** An ECOINM application is an instance of the ECOINM framework.

## 7.4 Merging Procedure

The procedure of merging applications in ECOIN, which is explained in more detail in [Kaleem 03] can be summarized as follows:

- Compare the context frames of the two applications

Example: Context frame of air ontology

{{af[47]:moneyAmount, {af:currency}},

{af:currency, {af:format}},

{af:airport, {af:format}},

{af:price, {af:currency, af:coverage, af:type}},

{af:date, {af:format, af:dateType}}}

Example: Context frame of car rental ontology (see Figure 7.6)

{{cr[48]:price, {cr:type, cr:period}},

{cr:city, {cr:symbol}},

{cr:date, {cr:format}}}

[47] af corresponds to the URI (Uniform Resource Identifier) for the airfare application
[48] cr corresponds to the URI (Uniform Resource Identifier) for the car rental application

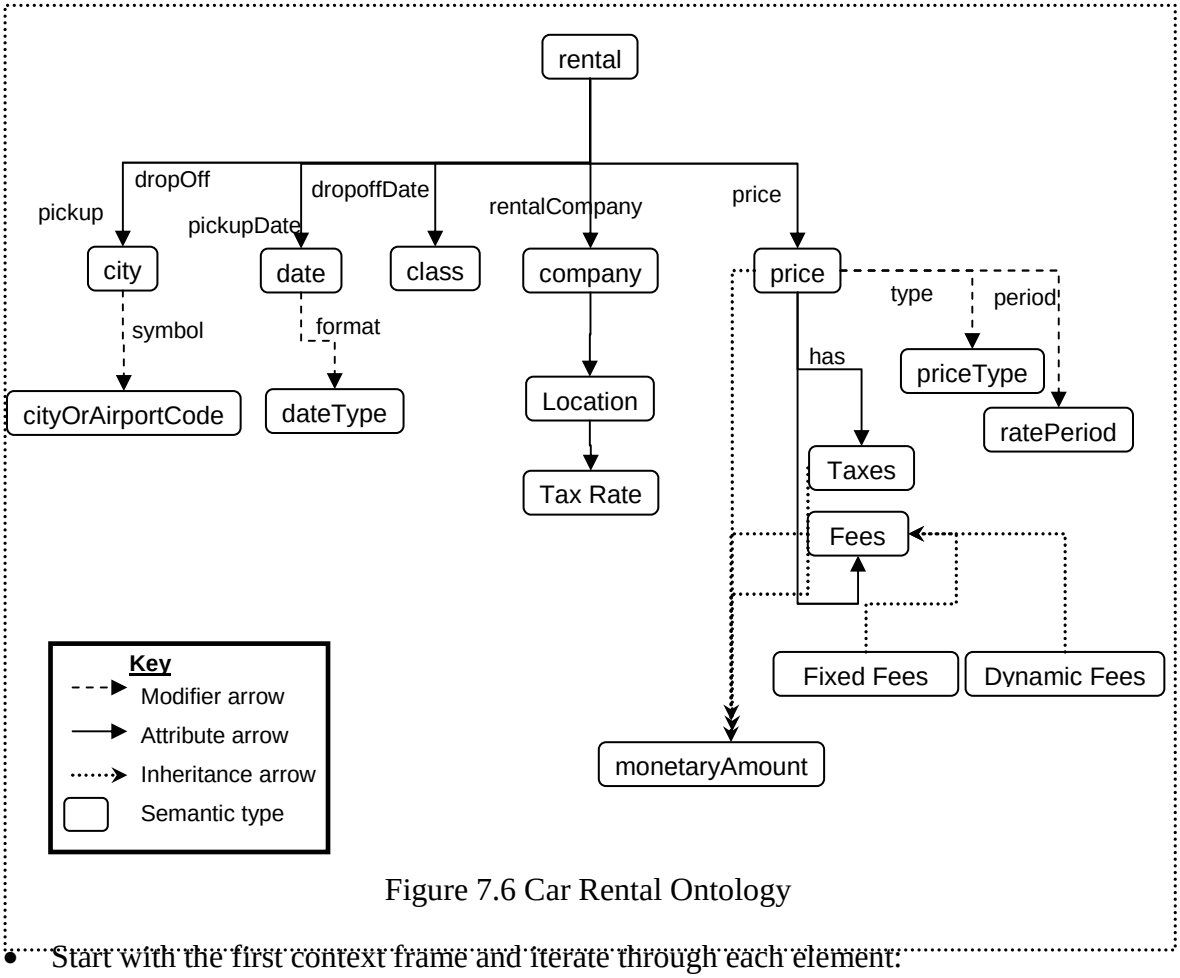



Figure 7.6 Car Rental Ontology

- Start with the first context frame and iterate through each element:

  e.g. i) {af:moneyAmount, {af:currency}},

  ii) {af:airport, {af:format}}

  1. If the semantic type has a corresponding type in the second ontology which can semantically have the same set of modifiers choose one of them to be upward inherited, or create a new type that can be related to both types.

     e.g i) af:moneyAmount corresponds to cr:monetaryAmount, select "af:moneyAmount"

     ii) create a type district that corresponds to both airport and city

     Note that

     - Different names for the same concept
       e.g. (moneyAmount vs. monetaryAmount)
     - Related but different concepts
       e.g. (revenues vs. profits)
     - More specialized or general versions of the same concept
       e.g. (financials vs. profits)

     can all qualify to have the same set of modifiers.

  2. Declare an *isomodifiertype(App$_1$, App$_2$, Term$_1$, Term$_2$)* relationship between the upward inherited or newly created semantic type and the related type(s),

.

which denotes that $Term_1$ in $App_1$ has compatible modifiers with $Term_2$ in $App_2$. e.g.

i) isomodifiertype(tr, cr, tr:moneyAmount, cr:monetaryAmount)
ii) isomodifiertype(tr, cr, tr:district, cr:city)
isomodifiertype(tr, af, tr:district, cr:airport)

Refer to Figure 7.7 for an illustration of the upward inheritance and isomodifiertype relationship.

3. For each modifier of the semantic type under consideration:
    - if there is a corresponding modifier defined in the related type, choose one of them to upward inherit

e.g. considering {cr:city, {cr:symbol}} and {af:airport, {af:format}}} choose cr:symbol for upward inheritance.

- declare an *isomodifier*(*$App_1$, $App_2$, $Term_1$, $Term_2$)* relationship between the upward inherited modifier and the related modifier, which means that $Term_1$ in $App_1$ is a compatible modifier with $Term_2$ in $App_2$.

e.g. isomodifier(tr, af, tr:district:symbol, af:airport:format)

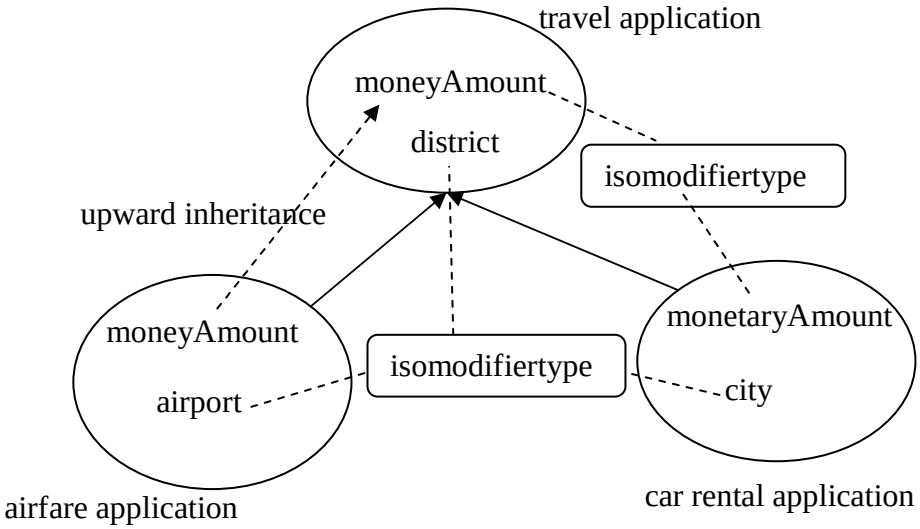



Figure 7.7 Upward Inheritance and Isomodifiertype Relationship

4. If there is a need for new modifiers because of the integration define them

e.g. date format modifier (e.g. European vs. American) may need a separator (e.g. "/" vs. "-" ) modifier. (See Figure 7.8 for an illustration)

- Continue with the second context frame and iterate through each element that has not been considered yet.
- Consider other ontology elements, which may need modifiers because of the merging.
- If an attribute definition is used in a conversion function in any of the applications to be merged or the merger application, relate that attribute name to the merger ontology

.

with *isoattribute*(*App*$_1$, *App*$_2$, *Term*$_1$, *Term*$_2$), which means that Term$_1$ in App$_1$ is a compatible attribute with Term$_2$ in App$_2$.

- If there are compatible (define compatibility somewhere) contexts between applications to be merged, relate context identifiers to the merger ontology with *isocontext*(*App*$_1$, *App*$_2$, *Term*$_1$, *Term*$_2$), which means that Term$_1$ in App$_1$ is a compatible context with Term$_2$ in App$_2$.
- Define new modifier values, new conversion functions, and new ontology terms in the new application if needed.

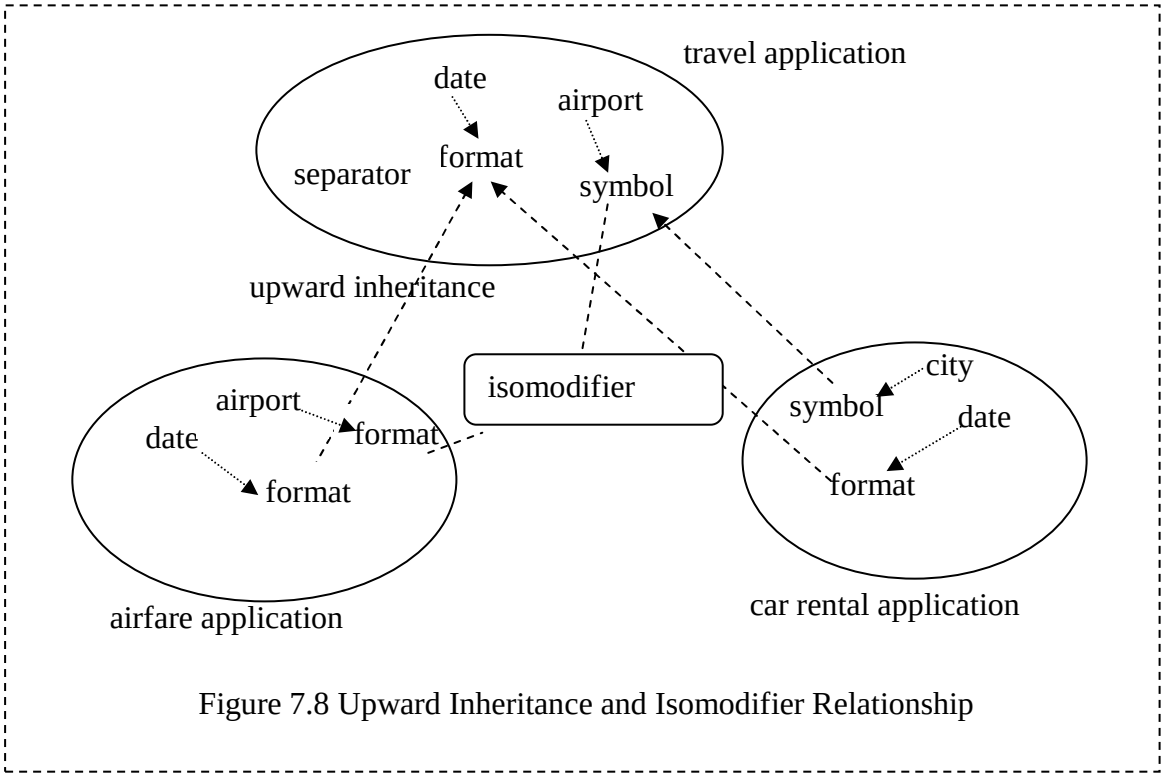



Figure 7.8 Upward Inheritance and Isomodifier Relationship

# Chapter 8

# The ECOIN Prototype

The ECOIN prototype provides an infrastructure for the realization of the COIN strategy through the ECOIN framework. The prototype consists of client, mediation and server processes as shown in Figure 8.1. Client processes refer to programs that aid the user in creating ECOIN applications, such as the textual application editor [Lee 03], or the CLAMP merging tool [Kaleem 03]; and in rerouting user queries to the mediator processes and answer sets to the users. Mediator processes rewrite user queries by utilizing application metadata to produce a mediated query and create and execute an optimized query plan. Server processes are those programs that allow access to traditional databases, web services and web pages.

In this chapter, we provide a high level description of the client and server processes and a more detailed description of the mediation processes. Software for the application metadata module is discussed in detail in [Lee 03] and [Kaleem 03], the latter focusing on merging ECOIN applications.

## 8.1 Client Processes

Client processes in ECOIN can be described under two categories: application creation and query formulation.

### 8.1.1 Application Creation

In the original COIN prototype, COIN applications were created using the COINL language which was then parsed and stored in an Eclipse prolog-based database. In the ECOIN prototype, ECOIN applications are stored in flat files as a set of First Order Logic (FOL) rules. In the most basic representation rule(H, B) is used to express the head H and the body B of a FOL rule. When the body is empty, B is replaced with true, and when there are more than one body clauses B takes the form of ($B_1$, …,$B_n$).

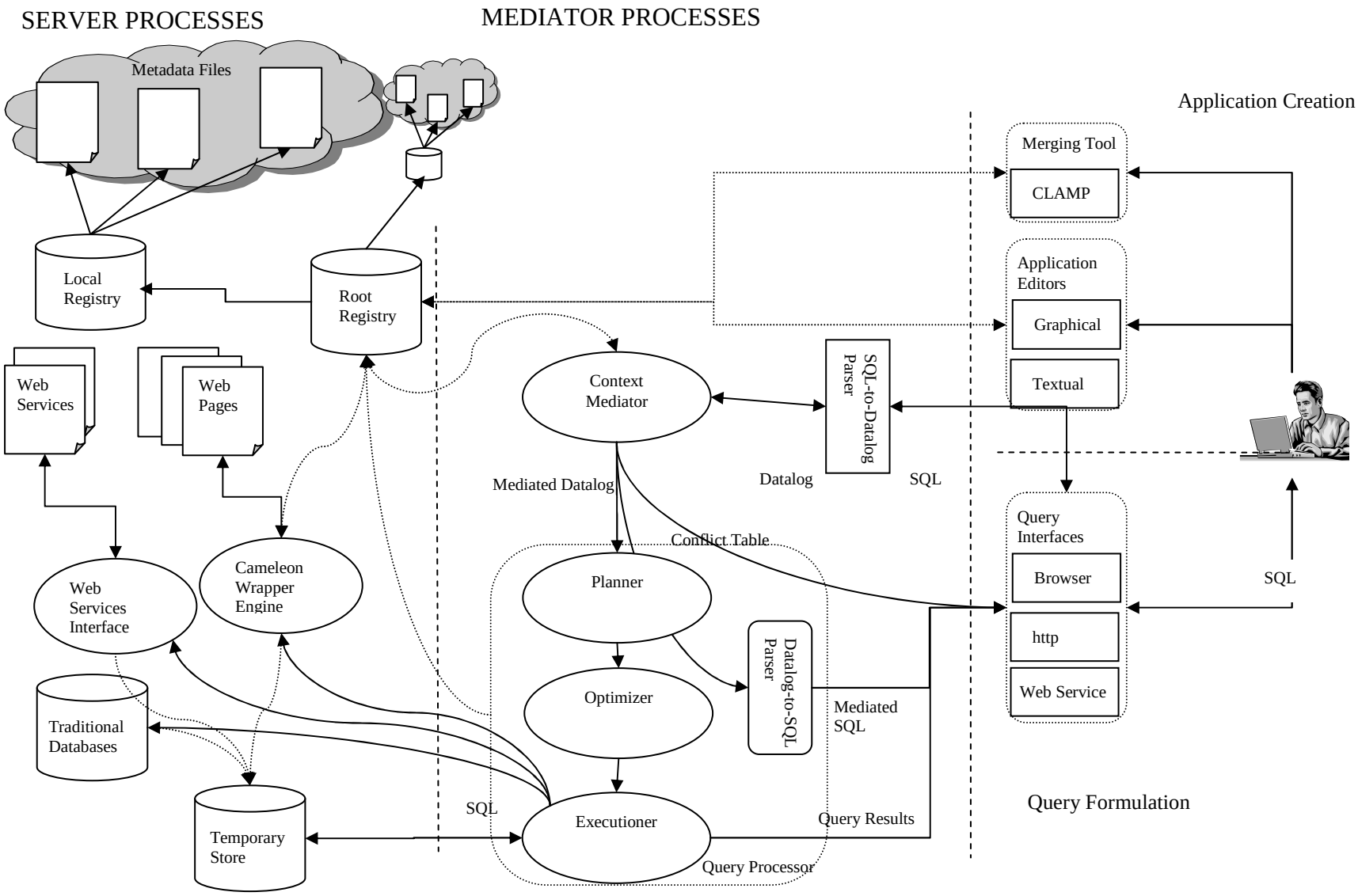



Figure 8.1 The ECOIN Prototype Architecture

.

For example, the dynamic modifier declaration from Chapter 5, shown below:

```
modifier(price, Price, currency, c_world, M) ←
        attribute(Price, product, Product),
        attribute(Product, country, Country),
        attribute(Country, officialCurrency, M).
```

would be expressed as a rule as follows:

```
rule(modifier(price, Price, currency, c_world, M),
        (attribute(Price, product, Product),
        attribute(Product, country, Country),
        attribute(Country, officialCurrency, M))).
```

By using XSLT, ECOIN application files can be converted into equivalent RDF, RuleML, and RFML representations as described in [Lee 03]. These transformations are syntactical in nature, and are aimed at making parsing easier for different programs in the ECOIN prototype as well as increasing the readability of ECOIN metadata by external users and programs.

ECOIN applications can be created either manually by directly entering ECOIN rules in a flat file or auto-generated through the use of an application editor tool. The primary application editor, shown in Figure 8.2., is a textual tool that allows users to generate ECOIN application rules with a simple point and click interface. Graphical application editor in its current state allows users to view ECOIN application rules graphically as shown in Figure 8.3.

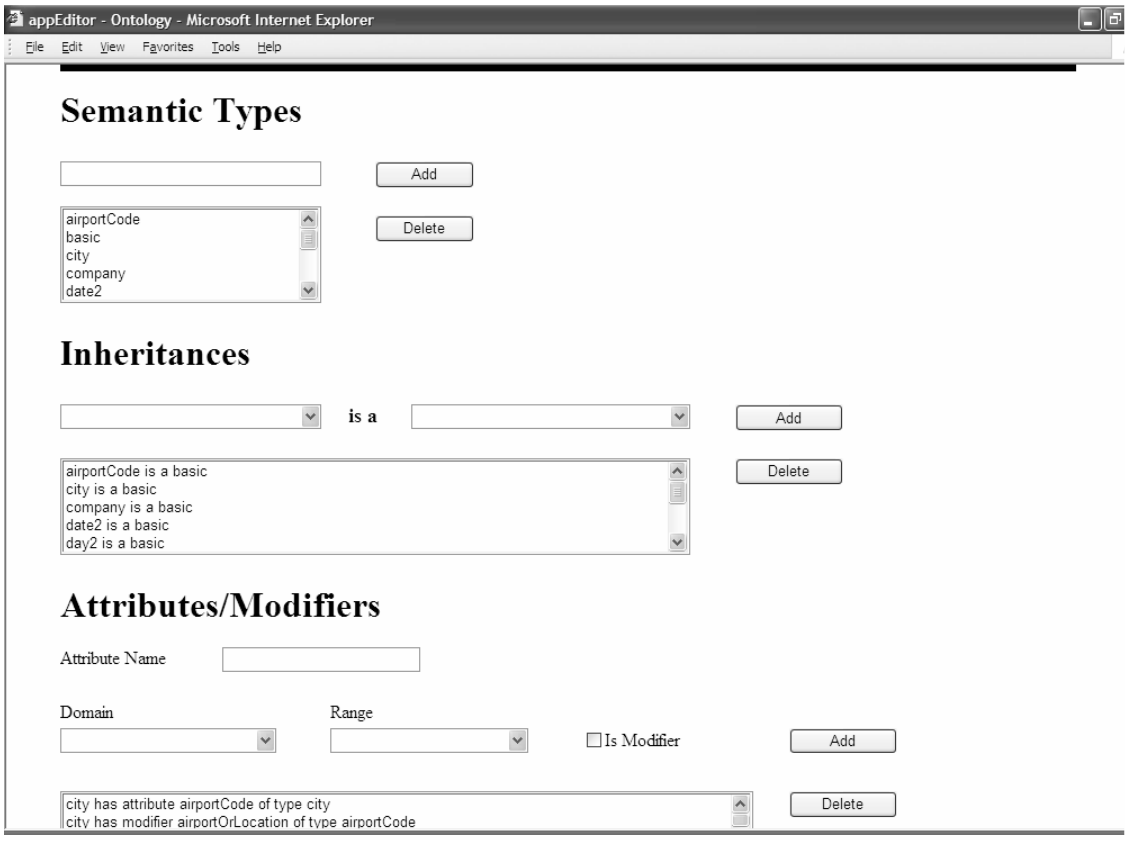



Figure 8.2 ECOIN Textual Application Editor

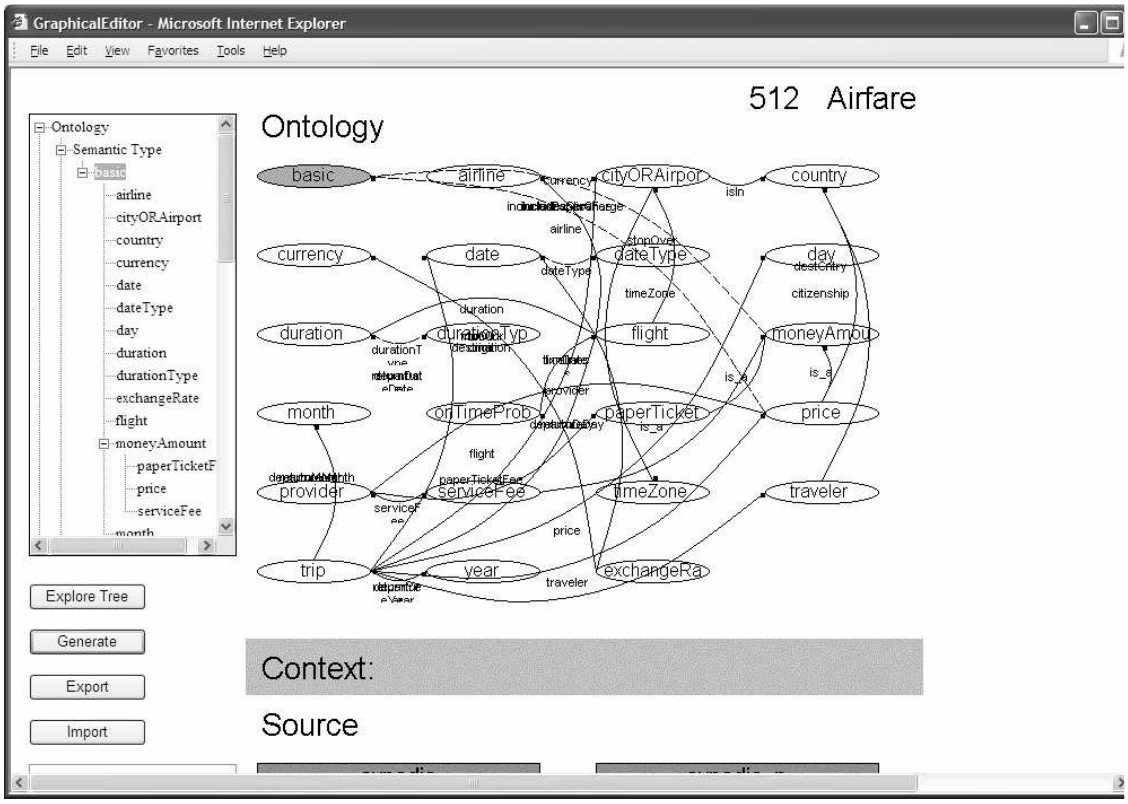



Figure 8.3 ECOIN Graphical Application Editor

CLAMP is a tool that aids users in merging disparate ECOIN applications. It guides the user by presenting modifiers of the applications to be merged and allows users to link semantic types, attributes and modifiers with a simple point and click interface.

## 8.1.2 Query Formulation

Queries to the ECOIN application are handled through the query interfaces that accept user queries in SQL and return answers in tables. The ECOIN Demo web interface shown in Figure 8.4, allows users to issue SQL queries with a receiver context and trace how it is processed by the mediation engine by going through SQL to Datalog translation, conflict detection, mediation, Datalog to SQL translation, and execution stages.

As shown in Figure 8.4, the user first inputs an SQL query in the SQL box, then chooses a receiver context, which specifies how the user expects the result set in terms of its semantics. The stage may be one of the six stages shown in Figure 8.4. Naïve Datalog and context sensitive stages display the Datalog equivalent of the input SQL query and its context adjusted form (i.e. well-formed query from Chapter 4) respectively. Conflict detection displays a matrix of the detected conflicts between the source and receiver contexts. Mediation stage outputs the mediated query which is a rewriting of the original query after detecting and reconciling conflicts between sources and the receiver. The SQL translation stage shows the SQL equivalent of the mediated query. Finally, the execution stage displays the results obtained from the data sources after executing the mediated query. The outputs are displayed in the Result box.

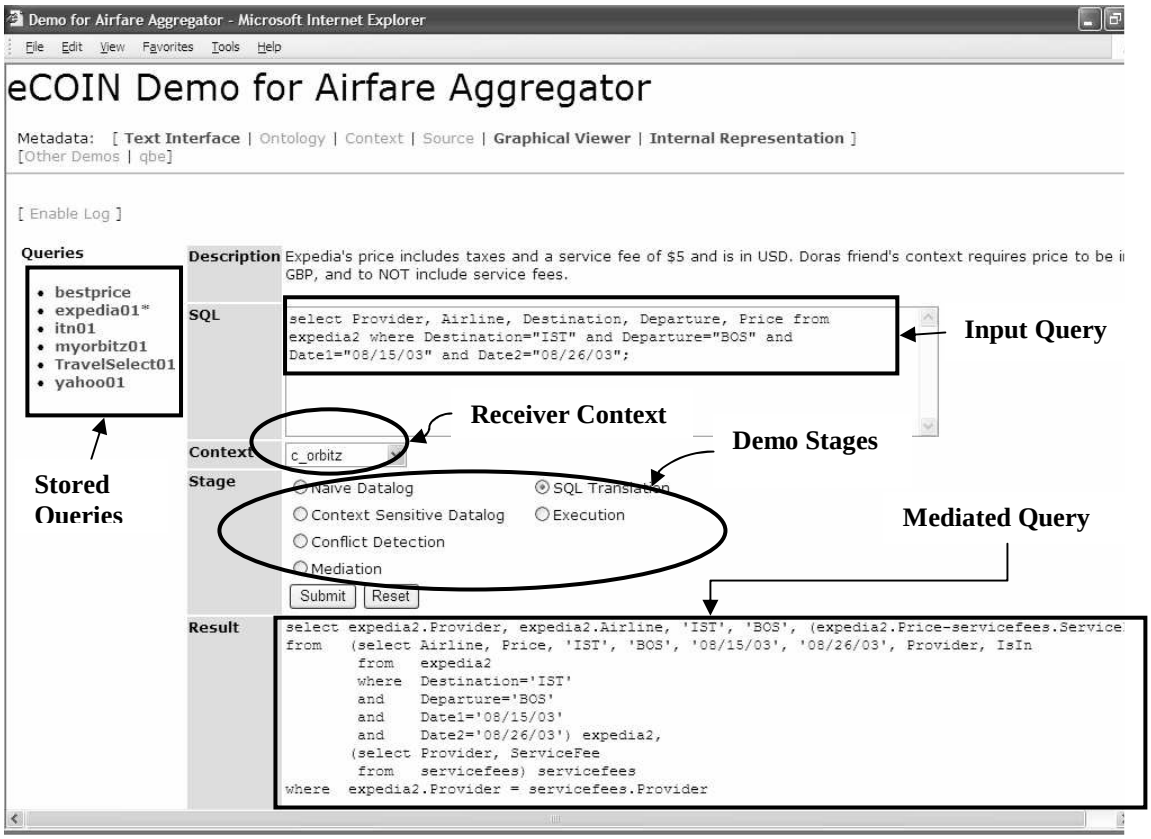



Figure 8.4 ECOIN Web Based Query Interface

Other query interfaces are designed for programming purposes and let programs to call the mediation engine with an input query and context issued against an application through the HTTP and SOAP protocols. These interfaces are designed to be used in user application programs.

# 8.2 Mediator Processes

The mediator processes consist of the context mediation engine, which accepts an SQL query and produces a mediated Datalog query, and the query processor with optimization and execution modules. The registry that stores or points to ECOIN application files and SQL to Datalog, and Datalog to SQL parsers are glue programs between the mediator and client processes.

The context mediation engine is implemented using Eclipse Prolog distributed by the ECRC Eclipse Prolog[49] is distinguished from other prolog implementations such as XSB Prolog[50] with its strong support for constraint logic programming.

## 8.2.1 The Abduction Engine

The abduction engine in ECOIN is an extended version of the COIN abduction engine. As in COIN, it takes the form of a meta-interpreter [Sterling and Shapiro 94]

[49] http://www-icparc.doc.ic.ac.uk/eclipse/

[50] http://www.cs.sunysb.edu/~sbprolog/xsb-page.html

extended with rules that incorporate the new features of ECOIN framework. The skeleton of the recursive abduction algorithm is shown in Figure 8.5.

```
1.  abductively_prove(true):-!.

2.  abductively_prove((H,B)) :-!,
abductively_prove(H),
abductively_prove(B).

3.  abductively_prove(Lit) :-
builtin(Lit), !,
call(Lit).

4.  abductively_prove(Lit) :-
builtin_clause(Lit, Body), !,
abductively_prove(Body).

5.  abductively_prove(Lit):-
abducible(Lit), !,
post_constraint(Lit).

6.  abductively_prove(Lit) :-
rule(Lit,Body),
abductively_prove(Body).
```

Figure 8.5 Skeleton of the meta-interpreter implementing the Abduction Engine

The above skeleton has the following declarative reading:

- Rule 1 corresponds to the base condition and states that the empty goal "true" is true.
- Rule 2 states that conjunctive goals H and B are true if both of them are true.
- Rule 3, states that if a goal is a built-in, it is true if its execution does not fail. Built-ins are either Prolog built-ins such as ground/1 or var/1; or user defined built-ins such as the constant semantic object builder cste(S, skolem(S, V, C, 1, cste(V)), C, V).
- Rule 4, states that built-in clauses are true if their body can be abductively proven. User defined clauses correspond to the basic definitions provided in the ECOIN framework. For example the context frame definition in Chapter 5:
  $M(basic) = \emptyset$.
  $i=1..n,\ m_i \in M(\tau) \leftarrow modifiers(\tau,[m_1,\ldots,m_n])$.
  $m \in M(\tau) \leftarrow is\_a(\tau,\tau_s),\ m \in M(\tau_s)$.
  becomes a user defined set of clauses in the implementation as follows:

```
contextFrame (basic,[]).
contextFrame (T,M) :- modifiers(T,M1), is_a(T,ST), contextFrame(ST,M2),
union(M1,M2,M)
```

- Rule 5, states that if the goal is abducible then it is posted in to the constraint store. A goal is abducible if it is one of the constraints in [ =,<,>,=<,>=,<>, is, +, -, *, /] or one of the designated external relations such as yahoo relation from the airfare example. Abducibles limit what statements can be used in expressing the mediated query.
  The constraint store is implemented using the Eclipse CHR library and consists of the basic inequalities for consistency checking of abducibles as well as the symbolic equation solving rules (See Figure 8.6 for a sample).

*Sample Inequalities*

```
built_in     @ X leq Y <=> ground(X),ground(Y) | X @=< Y.
reflexivity  @ X leq X <=> true.

antisymmetry @ X leq Y, Y leq X <=> X = Y.

transitivity @ X leq Y, Y leq Z ==> X \== Y, Y \== Z, X \== Z | X leq Z.

subsumption  @ X leq N \ X leq M <=> N@<M | true.
subsumption  @ M leq X \ N leq X <=> N@<M | true.
```

*Sample Symbolic Equation Solving Rules*

```
sum_ground  @ sum(X,Y,Z) <=> ground(X), ground(Y) | Z is X + Y, bound(Z).
sum_ground  @ sum(X,Y,Z) <=> ground(X), ground(Z) | Y is Z - X, bound(Y).
sum_ground  @ sum(X,Y,Z) <=> ground(Y), ground(Z) | X is Z - Y, bound(X).
```

Figure 8.6 Sample Inequalities and Symbolic Equation Solving Rules from the Constraint Store

- Rule 6, states that if the goal is a rule in the application declarations then the goal is true if the body of the rule can be abductively proven. For example given the following rule in the application file:

```
rule(modifier(price, Object, type, c_yahoo, Modifier),
      (cste(priceType, Modifier, c_yahoo, "nominal"))).
```

abductive proof of modifier(price, X, type, c_yahoo, M) would be reduced to the proof of abductively_prove(cste(priceType, M, c_yahoo, "nominal")) .

In Figure 8.7, shown in the following two pages, we provide a trace of the example query provided in Chapter 4 page ?. Starting with the well formed Datalog query WQ2:

```
answer(VAirline,VPrice) ←
        yahoo′(_,Airline′, Price′, _, DDate′, ADate′ , DCity′, _, ACity′),
        value(Price′, c_user, VPrice),
        value(Airline′,c_user, VAirline),
        value(DDate′,c_user, “01/06/03”),
```

→ abductively_prove(answer(VAirline,VPrice))
**Rule 5**
abducible(answer(VAirline, VPrice))
post_constraint(answer(VAirline, VPrice))

→ abductively_prove(yahoo′(_,Airline′, Price′, _, DDate′, ADate′ , DCity′, _, ACity′), value(Price′, c_user, VPrice),value(Airline′,c_user, VAirline), value(DDate′,c_user, "01/06/03"), value(ADate′,c_user, "01/08/03"), value(DCity, c_user, "Boston"), value(ACity, c_user, "Istanbul"))
**Rule 2**
→ abductively_prove(yahoo′(_,Airline′, Price′, _, DDate′, ADate′ , DCity′, _, ACity′)).
→ abductively_prove(value(Price′, c_user, VPrice))
→abductively_prove(value(Airline′,c_user, VAirline)).
→abductively_prove(value(DDate′,c_user, "01/06/03")).
→abductively_prove(value(ADate′,c_user, "01/08/03"))
→ abductively_prove (value(DCity′, c_user, "Boston")).
→abductively_prove(value(ACity′, c_user, "Istanbul"))
→ abductively_prove(yahoo′(_,Airline′, Price′, _, DDate′, ADate′ , DCity′, _, ACity′)).
**Rule 6**
rule(yahoo′(ID′,Airline′, Price′, Tax′, DepDate′, ArrDate′, DepCity′, CxnCountry′, ArrCity′) ,
(yahoo(ID,Airline,Price, Tax, DepDate, ArrDate,DepCity,CxnCountry, ArrCity))).
where
ID′ = skolem(flightID, ID, c_yahoo, 1, yahoo(ID,…, ArrCity)),
Airline′=skolem(airline, Airline, c_yahoo, 2, yahoo(ID,Airline,…, ArrCity)),
Price′=skolem(price, Price, c_yahoo, 3, yahoo(ID,Airline,Price,…, ArrCity)),
Tax′=skolem(tax, Tax, c_yahoo, 4, yahoo(ID,…,Price,Tax,…, ArrCity)),
DepDate′=skolem(date, DepDate, c_yahoo, 5, yahoo(ID,…,Tax, DepDate,…, ArrCity)),
ArrDate′=skolem(date, ArrDate, c_yahoo, 6, yahoo(ID,… ,DepDate, ArrDate,…, ArrCity)),
DepCity′=skolem(airport, DepCity, c_yahoo, 7, yahoo(ID,…,ArrDate, DepCity,…, ArrCity)),
CxnCountry′=skolem(country, CxnCountry, c_yahoo, 8, yahoo(ID,…,CxnCountry, ArrCity)),
ArrCity′=skolem(airport, ArrCity, c_yahoo, 9, yahoo(ID,…,CxnCountry, ArrCity))).

→ abductively_prove(yahoo(ID,Airline,Price, Tax, DepDate, ArrDate,DepCity,CxnCountry, ArrCity))
**Rule 5**
abducible(yahoo(ID,Airline,Price, Tax, DepDate, ArrDate,DepCity,CxnCountry, ArrCity))
post_constraint(yahoo(ID,Airline,Price, Tax, DepDate, ArrDate,DepCity,CxnCountry, ArrCity)).

→ abductively_prove(value(Price′, c_user, VPrice))
**Rule 4**
builtin_clause(value(Price′,c_user, VPrice), Body)
abductively_prove(Body).
where Body is
isa(Price′, S), sourceValue(Price′, Vsrc), all_contextFrame(S, L), allcvts(S, O, Vsrc, L, c_user, VPrice).
**Rule 4**
builtin_clause(isa(Price′, S), Body) [S resolves to type price]
builtin_clause(sourceValue(Price′, Vsrc), Body) [Vsrc resolves to Price in yahoo(ID,Airline,Price, Tax, DepDate, ArrDate,DepCity,CxnCountry, ArrCity)]
builtin_clause(all_contextFrame(price, L), Body) [ L resolves to [currency, type, coverage] after abductively proving:
contextFrame (basic,[]).
contextFrame (T,M) :- modifiers(T,M1), is_a(T,ST), contextFrame(ST,M2), union(M1,M2,M) ]
builtin_clause(allcvts(price, O, Price, [currency, type, coverage], c_user, VPrice))
allcvts applies conversion functions for each modifier
e.g. for currency the following conversion function definition goes through the abductive proof procedure
cvt (commutative, t, $t_p$, currency, $MV_s$, $MV_t$, $t_p'$) ←
exchangeRate′(Currency$_1$, Currency$_2$, Rate),
value(Currency$_1$, c, $MV_s$),
value(Currency$_2$, c, $MV_t$),
value(Rate, c, Rate$_p$),
multiply($t_p$ ,Rate$_p$, $t_p'$).

**Finally, the following constraints are posted to the constraint store.**

{ visafees("Transit, UK, Turkey", VF), currencyconvert("GBP","USD", ExchangeRate, "05/01/03"), sum(Price, T, PT), mul(ExchangeRate,VF,VFA), sum(PT,VFA,PTV),sub(PTVS,5,PTV), sub(Final, 20, PTVS), sum(20,5,SP), sub(Final, SP, FSP), div(RT,2,FSP), sum(RT,SP,VPrice)}
and CxnCountry is resolved to "Great Britain"
…continued in the next page:

→ abductively_prove(value(Airline′,c_user, VAirline))
**Rule 4**
builtin_clause(value(Airline′,c_user, VAirline), Body)
abductively_prove(Body).
where Body is
isa(Airline ′, S), sourceValue(Airline ′, Vsrc), all_contextFrame(S, L), allcvts(S, O, Vsrc, L, c_user, VAirline).
**…In this case no conversion is found to be necessary; therefore nothing is posted to the store. Only variable resolutions are reflected in the answer.**

→ abductively_prove(value(DDate′,c_user, "01/06/03")).
**Rule 4**
builtin_clause(value(DDate ′,c_user, "01/06/03"), Body)
abductively_prove(Body).
where Body is
isa(DDate ′, S), sourceValue(DDate ′, Vsrc), all_contextFrame(S, L), allcvts(S, O, Vsrc, L, c_user, "01/06/03").
**…In this case static conversion takes place and DepDate is resolved to "06/01/03"**

→ abductively_prove(value(ADate′,c_user, "01/08/03"))
**Rule 4**
builtin_clause(value(ADate ′,c_user, "01/08/03"), Body)
abductively_prove(Body).
where Body is
isa(ADate ′, S), sourceValue(ADate ′, Vsrc), all_contextFrame(S, L), allcvts(S, O, Vsrc, L, c_user, "01/08/03").
**…In this case static conversion takes place and ArrDate is resolved to "08/01/03"**

→ abductively_prove (value(DCity′, c_user, "Boston")).
**Rule 4**
builtin_clause(value(DCity ′,c_user, "Boston"), Body)
abductively_prove(Body).
where Body is
isa(DCity ′, S), sourceValue(DCity ′, Vsrc), all_contextFrame(S, L), allcvts(S, O, Vsrc, L, c_user, "Boston").
**…Eventually the following abducible obtained from conversion functions are posted to the constraint store:**
cityAirport("Boston", DepCity),

→abductively_prove(value(ACity′, c_user, "Istanbul"))
**Rule 4**
builtin_clause(value(ACity ′,c_user, "Istanbul"), Body)
abductively_prove(Body).
where Body is
isa(ACity ′, S), sourceValue(ACity ′, Vsrc), all_contextFrame(S, L), allcvts(S, O, Vsrc, L, c_user, "Istanbul").
**…Eventually the following abducible obtained from conversion functions are posted to the constraint store:**
cityAirport("Istanbul", DepCity),

After a second run and constraint processing the following set of abducibles are obtained as the answer:
{{ answer(VAirline,VPrice),
yahoo(I,VAirline, Price, T, "06/01/03", "08/01/03", $Airport_1$, "Great Britain", $Airport_2$),
visafees("Transit, UK, Turkey", VF),
cityAirport("Boston", $Airport_1$),
cityAirport("Istanbul", $Airport_2$),
currencyconvert("GBP","USD", ExchangeRate, "05/01/03"),
sum(Price, T, PT),
mul(ExchangeRate,VF,VFA),
sum(PT,VFA,PTV),
mul(PTV,2,RT),
sum(RT,25,VPrice).}
{ answer(VAirline,VPrice),
yahoo(I,VAirline, Price, T, "06/01/03", "08/01/03", $Airport_1$, Cxn, $Airport_2$),
Cxn <> "Great Britain",
cityAirport("Boston", $Airport_1$),
cityAirport("Istanbul", $Airport_2$),
sum(Price, T, PT),
mul(PT,2,RT),
sum(RT,25,VPrice).}}

Figure 8.7 Trace of WQ2 from Chapter 4

value(ADate′,c_user, “01/08/03”)
value(DCity, c_user, “Boston”)
value(ACity, c_user, “Istanbul”).

we show each rule that applies during the abduction phase. The final set of abducibles shown at the end of Figure 8.7 can be written as a Datalog query, which is then translated by the Datalog to SQL translator to MQ2 shown in Chapter 4.

## 8.2.2 Query Processor

The next destination of the mediated query is the query processor which consists of a query planner, optimizer and executioner as shown in Figure 8.8.

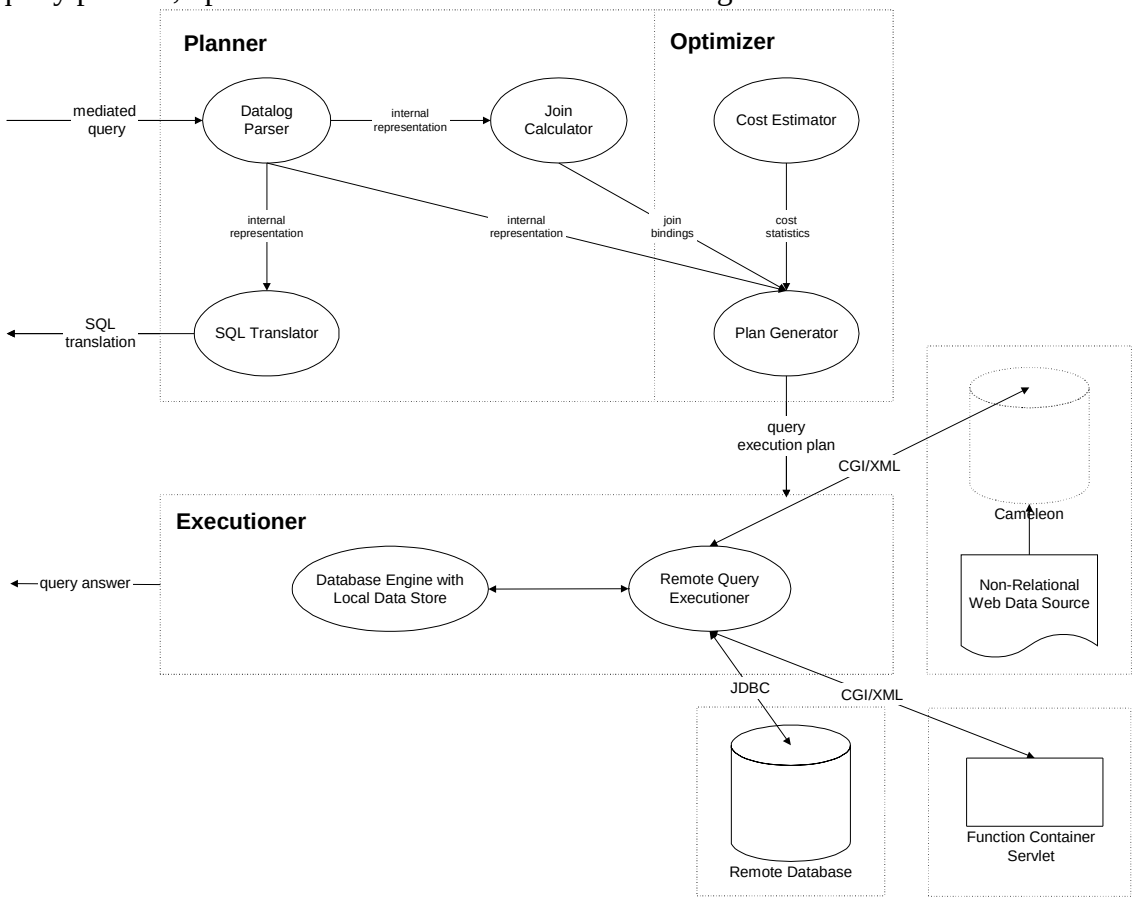



Figure 8.8 Architecture of ECOIN Query Processor (Adopted from Alatovic 02)

The Planner takes a Datalog query as an input and produces a query execution plan (QEP), which specifies constraints that need to be satisfied in the execution of component subqueries (CSQs). Optimizer uses cost estimates (transfer time of tuples across the network) to improve planner’s QEP by searching for an optimal execution path. The executioner dispatches the CSQs to the remote sources and combines the returned results. It also performs joins and condition filtering that could not have been done at the remote sources. Intermediate results are stored in the local data store, and the local RDBMS query processor is used to execute the final query over these intermediate results. More details can be found in [Alatovic 02]

## 8.3 Server Processes

Server processes are database gateways, wrappers for web pages and services, the local RDBMS used by the query processor and the distributed registry that stores pointers to metadata. Database gateways and wrappers provide a uniform way of accessing data sources by using a canonical query language such as SQL. Caméléon wrapper engine, described in Chapter 2, lets us treat web sites like limited traditional databases. Similarly, our web service wrapper lets us query web services using SQL with some restrictions.

The registry for the ECOIN system stores metadata needed by various applications in the prototype. The application editors use the registry to store and read ECOIN application metadata. The mediation engine use it to get the ECOIN application rules needed for the mediation. The query processor needs to obtain the schema information and location of data sources.

The registry has a distributed organization as shown in Figure 8.9 below:

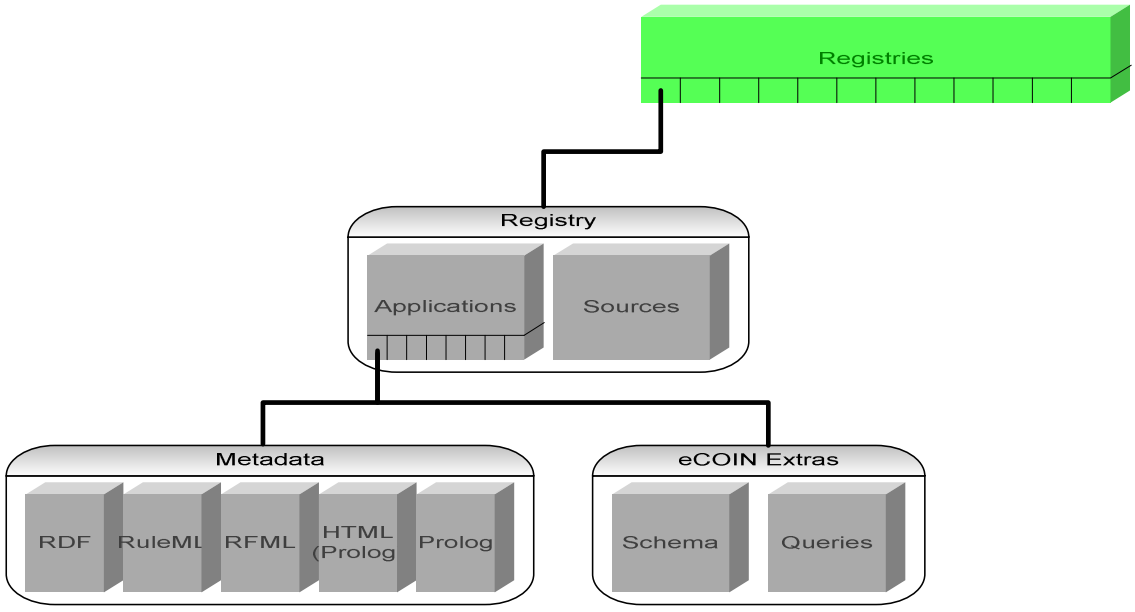



Figure 8.9 The Registry Organization in ECOIN (Adopted from [Lee 03])

The root registry is simply a set of pointers to local registries. ECOIN prototype maintained in our group at MIT, and the one located in Malaysia has different registries that are pointed from the root registries. Each local registry is organized by application identifiers and stores pointers to application metadata, which may be in a number of formats ranging from RDF to Prolog as shown in Figure 8.9. More details can be found in [Lee 03].

This completes the brief description of the ECOIN prototype, which is available online[51] to assist users from application creation to query execution. It is a powerful demonstration of the feasibility of the ideas described in this Thesis.

[51] http://context2.mit.edu/coin/

# Chapter 9

# ECOIN and the Semantic Web

In the previous chapters we laid out the ECOIN framework for achieving semantic interoperability among heterogeneous and autonomous data sources. In this chapter, we discuss the relationship between ECOIN and the Semantic Web (SWeb), the vision of achieving logical connectivity on the Internet [Berners-Lee 01]. The challenges of the SWeb will be similar to the challenges of ECOIN; therefore our research may offer many important lessons and approaches for the SWeb. At the same time, SWeb constitutes an important test bed for ECOIN, with its many heterogeneous and autonomous data sources. Our objective in this chapter is to provide some quick background on the SWeb, and point out some interesting future research directions.

## 9.1 The Semantic Web

The SWeb is the vision of achieving semantic interoperability on the Internet. The SWeb differs from older application environments in many ways, but particularly in its huge number of autonomous sources and the rapid and continuous change these sources are going through [Manola 02].

The SWeb introduces a set of layered standards to make data on the web well-defined for machines to reason with. These layered standards are often illustrated with the SWeb stack diagram shown in Figure 9.1. In the lowest layer, there are Uniform Resource Identifiers (***URI***) that identify resources on the web, and ***Unicode*** that encodes every character with a unique number independent of the platform, program, or language. ***XML*** provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documents. ***DTD*** (Document Type Definition)--not shown in the diagram-- is the grammar of an XML document that provides a list of the elements, attributes, comments, notes, and entities contained in the document as well as their relationship to one another within the document. XML ***Namespaces*** refer to collections of names, identified by URI references. ***XML Query*** language, based on ***XML Path*** expressions that address parts of XML documents, provides features for retrieving information from diverse XML sources. ***XML Schema*** is a language for restricting the structure of XML documents. ***RDF*** is a datamodel for objects ("resources") and relations

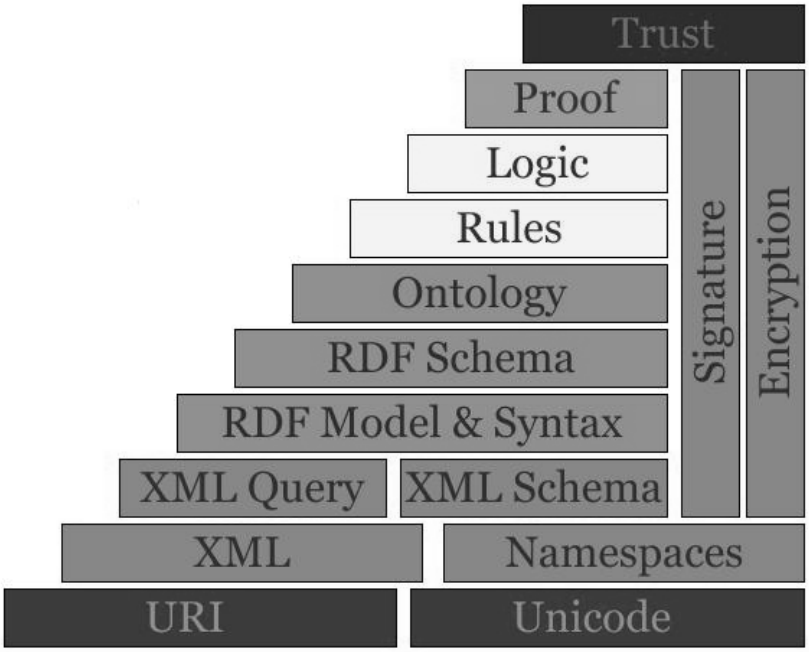



Figure 9.1 Semantic Web Stack

between them, and provides a simple semantics for datamodels that can be represented in an XML syntax. ***RDF Schema*** is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalization-hierarchies of such properties and classes. In the ***Ontology*** layer, ***OWL*** is a proposed ontology language that adds more vocabulary to RDFS for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes. So far main contributions of Semantic Web have been in the offering of standard languages for data and ontology representation. As shown in Figure 9.2, research at the ***Rules, Logic, Proof and Trust*** layers, is still in early phases and many of the issues surrounding the upper layers are relatively less understood and currently being investigated by many researchers. The current status of the SWeb is depicted in Figure 9.2: XML has been a standard since 1998 and is widely deployed to achieve interoperability within applications; Rules do not currently have a standard, although RuleML is being considered by the World Wide Web Consortium (W3C), who develops specifications, guidelines, software, and tools for the SWeb, etc.

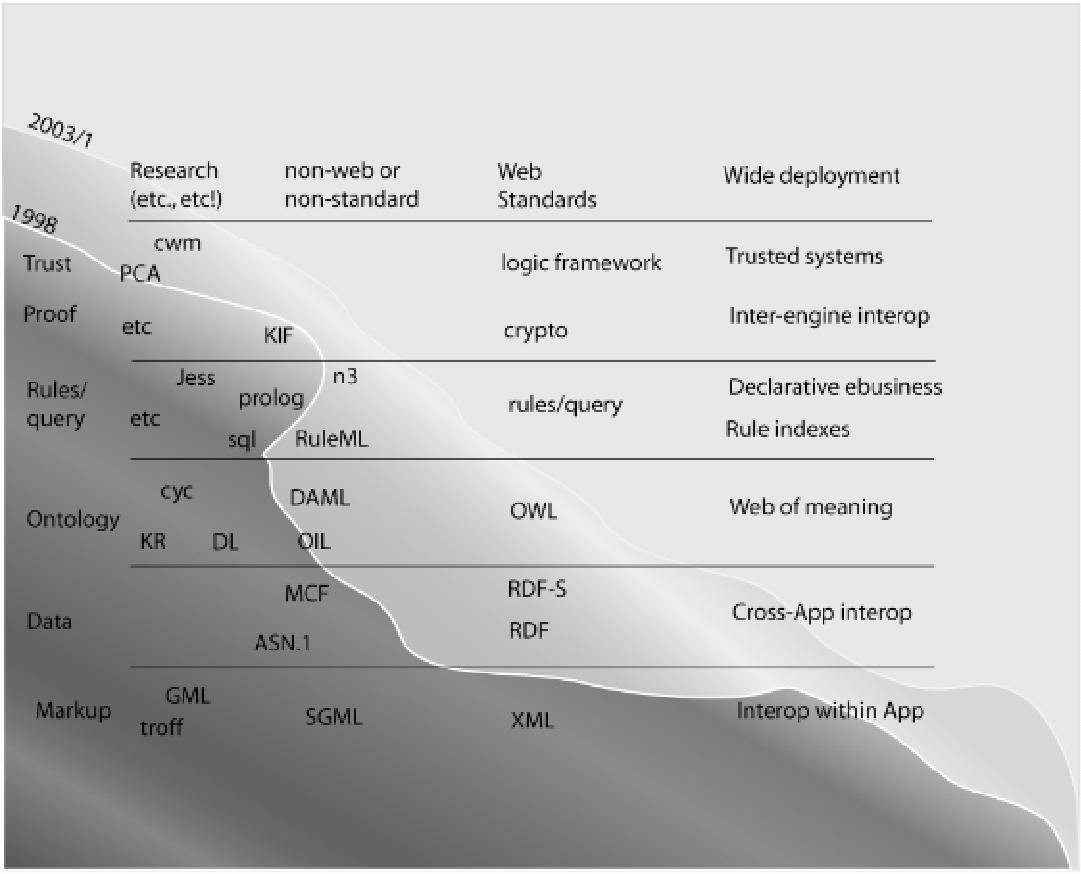



Figure 9.2 Semantic Web Status (adopted from [Berners-Lee 03])

# 9.2 The Semantic Web and Relational Databases

In this section, we aim to establish the relationship between the SWeb and the relational data model, the canonical representation of data sources in ECOIN. In the relational model, databases have tables, which are sets of rows. Each row is a collection of data cells identified with a field (column) name. Relational schemas define the names and domains of fields, as well as a set of integrity constraints including key constraints, and referential integrity constraints.

Although XML, and its schema definition languages DTD, and XML Schema can be used to express relational data models, the Semantic Web data model supports the relational data model mainly through the use of Resource Description Framework (RDF) and its corresponding schema language RDFS. RDF is preferred over XML, because it unifies set of all possible XML representations of a fact into one statement. RDF can be thought of as the XML encoding of a relational table cell as shown in Figure 9.3 below. In similar vein, RDF schema corresponds to the schema of a relational table.

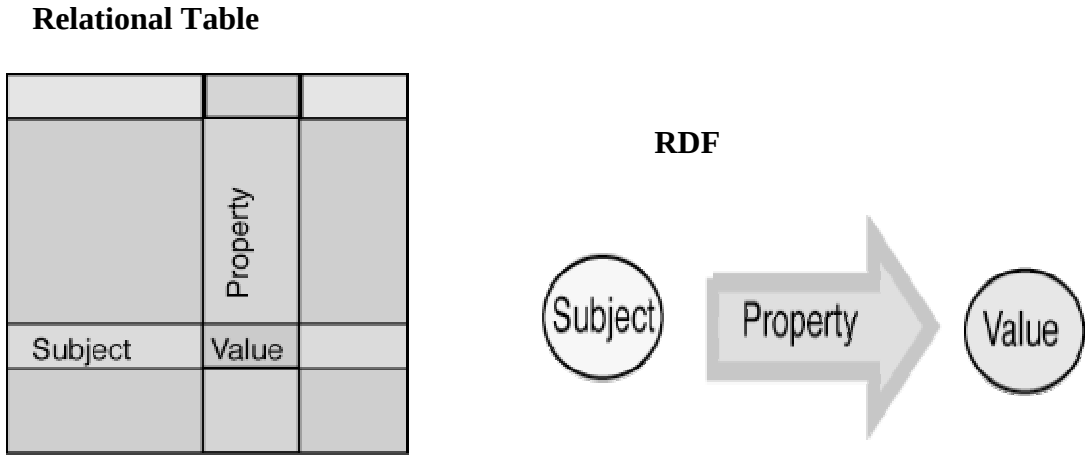



Figure 9.3 RDF vs. Relational Model (adapted from [Tim Berners-Lee 03])

RDF has a simple data model consisting of triples with the following components (see Figure 9.3)

- a *property* that describes some *relationship* (also called a predicate),
- a *value* that is the *subject* of the statement, and
- a *value* that is the *object* of the statement.

A property must be a URI reference, whereas the subject and object may be a blank node (a node without a URI), a constant or a URI reference. Set of RDF triples constitutes an RDF graph, in which subjects and objects are called the nodes.

A row in the relational model can be expressed as a set of RDF nodes, with the following mapping:

- a blank RDF node corresponding to a row, with its rdf:type property corresponding to the table name;
- a set of RDF properties corresponding to the column names; and,
- a set of constant values for each property corresponding to data cells.

A row in yahoo relation from the airfare example for instance could be pictured as a graph as shown in Figure 9.4. In this figure, the blank node corresponds to a row from the yahoo relation. The rdf:type property of the blank node is set to the relation name yahoo in the sense that a row is an instance of the predicate corresponding to the table name. The column names define the outgoing property arrows from the blank node, which points to constant values for each property corresponding to data cells in the yahoo relation.

RDFS is the vocabulary description language for RDF. In terms of its expressive power it is comparable to Entity Relation models, and can be used to describe database schemas. The schema for the yahoo relation from the airfare example can be defined as shown in Figure 9.5. In the Figure, all column names are defined as instances of class

rdf:Class, and their domains are restricted by the rdf:datatype property. Unlike, database schemas, however, RDFS restrictions are not automatically enforced. It is left to the individual programs to process these constraints. Furthermore, RDFS does not provide built in support to express most integrity constraints such as the key and uniqueness

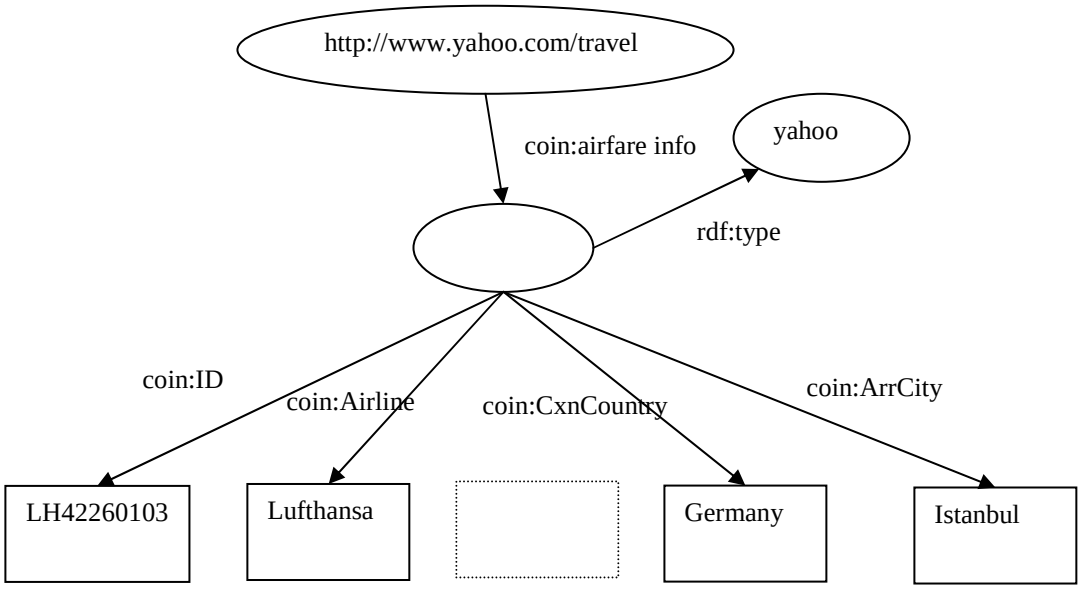



Figure 9.4 A row from airfare relation yahoo in RDF data model

constraints. Expression of these constraints is left to higher level languages such as OWL, which builds on RDFS bare minimums.

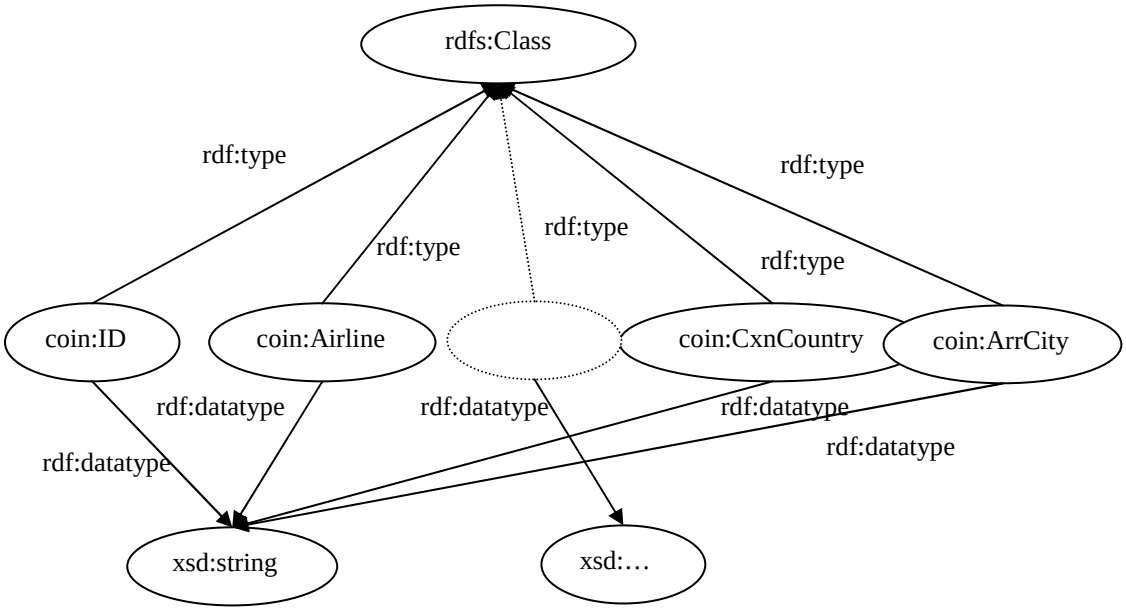



Figure 9.5 Schema of airfare relation yahoo in RDF Schema

## 9.3 The Semantic Web and Ontologies

Web ontology language (OWL), which is a standard candidate for the Semantic Web, is an outcome of web ontology work that started in mid-nineties with projects such as SHOE [Luke et al. 97], Ontobroker [Decker et al. 98], OIL [Horrocks et al. 00], and DAML+OIL [Connoly et al. 01]. OWL is built on RDF(S), and is encoded and written as an RDF graph. It has three species or versions: OWL Lite, OWL DL (where DL stands for "Description Logic"), OWL Full. OWL Lite, is the most basic version that allows the expression of classification hierarchy and simple constraints (e.g. cardinality constraints of 0 or 1). OWL DL provides maximum expressiveness while being computationally complete (all conclusions can be computed) and decidable (all computations can be performed in finite time). OWL Full is the most expressive version of the language without having any computational guarantees like OWL DL.
OWL, compared to ECOIN ontology language, (with constructs semantic type, attribute, is-a, and modifier) has a richer set of ontology constructs. Mappings between the ECOIN and OWL ontologies, with the exception of modifiers, can be ordinarily established as shown in Table 9.1:

| **ECOIN** | **OWL** |
|---|---|
| Semantic Type | Class |
| Attribute | ObjectProperty |
| is-a | subClassOf |

Table 9.1 ECOIN to Semantic Web Mapping

The semantic type concept in ECOIN corresponds to the Class concept as *a class identifier* in OWL. Syntactically, the semantic type *Trip* from the airfare example would be represented in OWL as follows:

```
<owl:Class rdf:ID="Trip"/>
```

The attribute concept in ECOIN corresponds to the concept of Property in OWL. In ECOIN, domain and range of a property are not enforced, and can be determined at run time. Similarly, OWL allows domain and range values to be determined before run time, and it also provides the flexibility to leave them undefined. The destination attribute declaration, attribute(trip, destination, airport), would be defined in OWL as follows:

```
<owl:ObjectProperty rdf:ID="destination">
  <rdfs:domain rdf:resource="#Trip" />
  <rdfs:range  rdf:resource="#Airport" />
</owl:ObjectProperty>
```

The is-a relationship between semantic types in ECOIN are represented by the subClassOf relationship in OWL. The is_a(price, moneyAmount) from ECOIN would be represented in OWL as:

```
<owl:Class rdf:ID="price">
  <rdfs:subClassOf rdf:resource="#moneyAmount" />
</owl:Class>
```

The concept of a modifier, however, do not have a direct counterpart in OWL. Since a modifier is a special type of attribute, it can be represented as a an attribute in OWL, and

annotated as a modifier with a property. For example, the currency modifier for moneyAmount semanticType could be represented in OWL as shown in Figure 9.6.

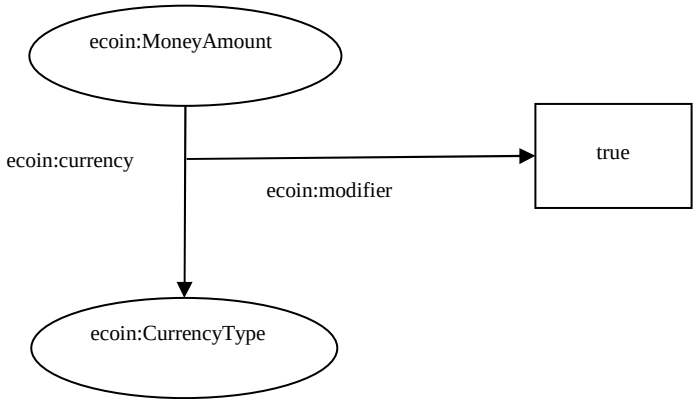



Figure 9.6 Modifier Representation in OWL

In Figure 9.6, a modifier property is defined from a propery to a constant. The modifier property acts like a flag that designates whether a given property is a modifier property.

Perhaps, a better way to incorporate modifiers would be to create a subclass of owl:ObjectProperty like the way owl:SymmetricProperty is derived and use that property in modifier declarations. In this case the following declarations can be used, which more closely corresponds to the ECOIN data model:

```
<owl:Class rdf:ID="ModifierProperty">
  <rdfs:subClassOf rdf:resource="owl:ObjectProperty" />
</owl:Class>

<owl:ModifierProperty rdf:ID="currency">
  <rdfs:domain rdf:resource="ecoin:MoneyAmount" />
  <rdfs:range  rdf:resource="ecoin:CurrencyType" />
</owl:ObjectProperty>
```

If we were to redraw the airfare ontology we have shown in Chapter 4 using OWL, it would almost be an identical graph with the exception of exchanging modifier arrows with ModifierProperty arrows, and is-a arrows with the subClassOf property arrows.

# 9.4 The Semantic Web and Context

The issue of context has been discussed in the Semantic Web community after suggestions that the idea of contexts were missing in RDF. The issue has been considered under the subjects of reification, and *Notation 3*[52] (N3)'s implementation of contexts as a container set. But a decision to include contexts in RDF standard has not been made yet.

[52]Simplified RDF language

Reification deals with the situation of making statements about statements. Reification relates to the idea of contexts because contextual statements can be thought of as statements about the truth of a statement in a context (i.e. in McCarthy's notation ist(c, statement)). For example, the following statement $S_1$ about the price of an airfare:

($S_1$) trip LH425060103 has a price whose value is 550

could be expressed as a triple:

[ecoin:LH425060103 ecoin:price "550"]

and reified as follows:

[[ecoin:LH425060103 ecoin:price "550"] ecoin:ist contexts:c]

In RDF, reification is implemented with the introduction of a new type rdf:Statement which has the properties of rdf:subject, rdf:predicate, and rdf:object. Subjects that refer to objects with the rdf:Statement type, different from other types, refer to the whole statement which is the combination of subject, predicate and property properties.

It has also been proposed in the RDF discussion groups that RDF be turned into a quadruple to include a context or statement id as a tuple. With this proposition $S_1$ could be represented as a quadruple:

[$S_1$ ecoin:LH425060103 ecoin:price "550"]

and $S_1$ could either be perceived as a statement or context id.

In yet another proposal to implement the notion of context [Klyne 00] in RDF, inspired by the N3 language, a new container class called *rdfc*:StatementSet is introduced to represent a collection of reified RDF statements. Context is then defined as a sub class of this container class and several properties such as asserts, assumes are used to add statements to the container, and to define relationships between contexts, etc. This proposal, like others, also has not found its way into the RDF standard yet.

## 9.5 The Semantic Web and Rules

The concepts of ontology, context and rules are fundamental in creating a semantic organization of knowledge. Ontologies are important in specifying the explicit semantics of data in the form of concepts and their relationships; contexts facilitate meaningful exchange of data with implicit semantics. Rules, on the other hand, are critical in expressing generalizable knowledge through the use of ontologies and contexts. In the ECOIN framework, for example, rules are used to express mappings between sources and the ontology, intensional expression of modifier values (i.e. context axioms), conversion functions that map object values between different contexts, and integrity constraints for

sources. Rules in ECOIN are closely tied with ontologies and contexts since they refer to constructs in the ontology and context identifiers.

Rules on the SWeb has recently gained more recognition and been included in the SWeb stack diagram on top of the Ontology layer. Currently, the most prominent effort in representing rules for the Semantic Web is the XML encoded Rule Mark up Language (RuleML) [Grosof 01]. RuleML aims to define a shared language that permits "both forward (bottom-up) and backward (top-down) rules in XML for deduction, rewriting, and further inferential-transformational tasks" [Boley 01].

ECOIN knowledge, such as elevation axioms, modifier declarations, conversion functions and integrity functions are rules that can be expressed on the SWeb using a standard rule language that supports ontologies like OWL. With the emergence of such a rule language standard on top of ontologies, these mappings will be clearer.

## 9.6 The Semantic Web and Logic Programming

The relationship between logic programming and the Semantic Web has been examined by analyzing the mappings between logic programming structures and XML and RDF [Boley 00]. Accordingly, basic RDF can be formalized with ground binary Datalog Horn facts. For example, the triple

[ecoin:LH425060103 ecoin:price "550"]

can be encoded as the ground Datalog Horn binary fact as:

ecoin:price (ecoin:LH425060103, "550")

Furthermore, RDF container structures such as bags can be transformed into lists in logic programming. Reification can be treated with the use of modal-logic (e.g. with the use of a belief operator), and the use of logic variables may enable the expression of rules using RDF.

With these mappings, RDF can be considered as a special case of knowledge representation with logic programming. This would then bring the possibility of using non-monotonic reasoning techniques employed in ECOIN, specifically abduction and constraint logic programming, in the context of Semantic Web. We leave the details of such a possibility for future work.

## 9.7 Future Work

There are several interesting directions for future work on gradually extending ECOIN approach to the SWeb. Some of these can be listed as follows:

- Extending Caméléon wrapper engine with OWL support and RDF output capabilities
- Using RDF documents as data sources
- Using OWL and (RDF/)RDFS as data schemas
- Using RuleML encoded rules to express elevation axioms and conversion functions
- Mediation of XML Query language based queries
- Investigating the representation of contexts on the SWeb

# Chapter 10

# Conclusion

In this Thesis, we addressed the two intertwined problems of *logical connectivity*, namely *data extraction* and *data interpretation* in the domain of heterogeneous information systems.

We, first, described the design and implementation of a general purpose, regular expression based Caméléon wrapper engine with an integrated capabilities-aware planner/optimizer/executioner, and IWrap semi-automatic wrapper generator. Compared with other existing approaches in the academia and industry, Caméléon and its accompanying tools provide a fine *balance of expressiveness and simplicity* in the data extraction domain.

Then, we provided a conceptualization for the dimensions of semantic heterogeneity, to better explain the nature of problems related to *data interpretation*. After presenting a brief analysis of semantic conflicts in financial information systems, we introduced three dimensions of semantic heterogeneity: *contextual, ontological,* and *temporal*. Furthermore, we defined a subcategory under ontological heterogeneities that referred to the heterogeneity in the way data items are calculated from other data items in terms of definitional equations as *equational ontological conflicts.*

Before describing the Extended Context Interchange (ECOIN) approach to achieving semantic interoperability among heterogeneous and autonomous data sources, we summarized the *Context Interchange* strategy employed in our predecessor COIN. COIN was built on the ideas of *contexts* [McCarthy 93], *heterogeneous database integration*, *abductive logic programming* [Kakas 00], and *deductive object-oriented data models* and provided a framework for addressing *contextual heterogeneities*. With ECOIN, we introduced a way to handle equational ontological conflicts by representing them as contextual heterogeneities via the existing representational framework of COIN with some minor changes. The reasoning framework, however, needed to be extended with *constraint logic programming* to enable reasoning with symbolic equations. This new intertwined reasoning framework is known as *abductive constraint logic programming* in the literature and has been successfully used in ECOIN as a *meta-interpreter* with symbolic equation solving capabilities.

Compared with existing tightly and loosely-coupled approaches in the literature, our approach provides a middle ground by combining the strengths of both approaches. Like

tightly coupled approaches (e.g. Pegasus [Ahmed et al. 91], InfoMaster [Duschka and Genesereth 97], Information Manifold [Levy 98]) we automate the task of rewriting a user query thus freeing the users from having to know the semantic details of sources; and like loosely coupled approaches (e.g. MRDSM [Litwin and Abdellatif 87] , VIP-MDBMS [Kuhn and Ludwig 88], TSIMMIS [Garcia-Molina et al. 95]) we enable users to dynamically choose how to receive data from sources thereby offering a level of flexibility unseen in tightly coupled systems.

With ECOIN, we also ventured into merging disparate ECOIN applications, which involves merging disparate ontologies and contextual knowledge. Our virtual and context-based approach to merging of ontologies is a hybrid of classical merging and alignment approaches. Like classical merging we produce a new ontology from a number of ontologies, (albeit a virtual one), and like ontology alignment approaches we use articulation axioms to relate the terms in disparate ontologies. Ease of merging disparate ECOIN applications demonstrate the scalability and extendibility of our approach.

## 10.1 Future Work

The completion of this Thesis also opens up many other research issues, which we hope to explore in the future. In this section we would like to mention a few of promising research areas.

First, representation frameworks used in both COIN and ECOIN are limited to representing contextual heterogeneities. While we were able to represent some of the ontological heterogeneities as contextual heterogeneities, it should also be possible to deal with them at the ontological level. For example, general relationships such as “Profit = Revenue – Expenses” can be represented at the ontological level as well. What representational extensions are needed to relate ontological constructs that are clearly distinct (e.g. Profit vs. Expenses) but somewhat related at the ontological level? When is it appropriate to represent ontological heterogeneities at the contextual level? These are questions that need to be addressed in future research.

We also left out issues related to temporal heterogeneities in this Thesis. As we mentioned in Chapter 3, temporal heterogeneities are orthogonal to both contextual and ontological heterogeneities, which suggests a different way to represent and reason with them. The ability to represent temporal heterogeneities without destroying our current framework would be an important research goal in the coming years.

In Chapter 9, we discussed the relationship between the Semantic Web and ECOIN and pointed out interesting synergies they exhibit. One of our research goals will be generalizing the ECOIN approach to the Semantic Web, which alters some of the fundamental assumptions of the semantic interoperability problem.

Finally, we would like to explore bio-informatics as a fertile field to apply our results and test the viability of our solutions.

# References


Abiteboul S. (1997). Querying semi-structured data. In Proceedings of ICDT, Jan 1997

Akman, V. Surav, M. (1997). The Use of Situation Theory in Context Modeling, Computational Intelligence 13(3): 427-438.

Akman, V. (2000). Rethinking context as a social construct, Journal of Pragmatics, 32(6):743-759.

Allen, C. (1997). WIDL Application Integration with XML, http://www.xml.com/pub/w3j/s3.allen.html

Ambrose, R (1998) A lightweight multi-database execution engine, Thesis (S.M.) Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science.

Apt, K.R. (1990). Logic Programming. Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B) 1990: 493-574.

Apt, K.R., Bol, R. (1994). ``Logic programming and negation: a survey", Journal of Logic Programming 19-20, pp. 9-71, (1994).

Ahmed, R., De Smedt, P., Du, W., Kent, W., Ketabchi, M., Litwin, W.,, Rafii, A.,,Shan, M. (1991). The Pegasus Heterogeneous Multidatabase System. IEEE Computer 24(12): 19-27.

Arens, Y., Knoblock, C., Shen, W. (1996). Query Reformulation for Dynamic Information Integration. Journal of Intelligent Information Systems 6(2/3): 99-130.

Baral, C., Gelfond, M. (1994). "Logic Programming and Knowledge Representation", Journal of Logic Programming, 19,20:73-148.

Barish, G., DiPasquo, D., Knoblock, C., Minton, S. (2000) A Dataflow Approach to Agent-based Information Management. IC-AI 2000. Las Vegas, NV.

Batini, C., Lenzerini, M., Navathe, S. B. (1986). A Comparative Analysis of Methodologies for Database Schema Integration, ACM Computing Surveys 18(4): 323-364.

Baumgartner, R., Flesca, S., Gottlob, G. (2001). "Declarative Information Extraction, Web Crawling, and Recursive Wrapping with Lixto". In Proc. LPNMR'01, Vienna, Austria, 2001.

Bressan, S., Bonnet, P. (1997). Extraction and Integration of Data from Semi-structured Documents into Business Applications, Conference on the Industrial Applications of Prolog, 1997.

Berners-Lee, T., Hendler, J., Lassila, O. (2001). The Semantic Web, Scientific American May 2001 Issue.

Berners-Lee,T.(2003). Standards, Semantics and Survival http://www.w3.org/2003/Talks/01-siia-tbl/slide8-0.html, Software and Information Industry Association (SIIA) meeeting (NYC, NY, USA).

Boley, H. (2001). Harold Boley: The Rule Markup Language: RDF-XML Data Model, XML Schema Hierarchy, and XSL Transformations, http://www.dfki.uni-kl.de/~boley/ruleml-mht.pdf , Invited Talk, International Conference on Applications of Prolog, Tokyo,

Boley, H. (2000). Relationships between Logic Programming and RDF. Pacific Rim International Conference on Artificial Intelligence : 201-218

Bouquet P., Ghidini C., Giunchiglia F., Blanzieri E. (2001). Theories And Uses Of Context In Knowledge Representation And Reasoning, Technical Report # 0110-28, Istituto Trentino di Cultura.

Bouquet P., Serafini L. (2003). On the Difference between Bridge Rules and Lifting Axioms. Modeling and Using Context, 4th International and Interdisciplinary Conference, CONTEXT: 80-93

Breitbart, Y., Tieman, L. (1984). ADDS - Heterogeneous Distributed Database System, Proceedings of the Third International Seminar on Distributed Data Sharing Systems, 7-24.

Burchert, H.-J. (1994). A resolution principle for constrained logics. Artificial Intelligence conclusions 66.

Bunge M. (1974). Semantics I: Sense and Reference, Vol. 1 of Treatise on Basic Philosophy. Dordrecht: Reidel.

Busse, S., Kutsch R. D., Leser, U. (1999). Federated Information Systems: Concepts, Terminology and Architectures, Technical Report Nr. 99-9, TU Berlin

Buvac, S., Mason,I. A. (1993) Propositional Logic of Context. In Proceedings of Eleventh Annual National Conference on Artificial Intelligence AAAI'93, pages 412-419.

Buvac, S. (1998). Contextual Information Integration, Ph.D. Thesis, Stanford University.

Castro L, Warren D. L. (2000). On the Computational Integration of Well-Founded and Stable Model Semantics, http://www.cs.sunysb.edu/~luis/rpe.ps.gz.

Chalupsky, H. (2000). OntoMorph: a translation system for symbolic knowledge. In Proceedings of Seventh International Conference on Knowledge Representation and Reasoning, pages 471--482, San Francisco, California, Morgan Kaufmann

Clark, K. (1978). Negation as failure. In H. Gallaire and J. Minker, ed.'s, Logic and Data Bases, p. 293--322. Plenum Press.

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

Connolly, D., van Harmelen, F., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., Stein, L. A. (2001). DAML+OIL (March 2001) reference description. W3C note 18 December 2001. http://www.w3.org/TR/2001/NOTE-daml+oil-reference-20011218

Dantsin, E., Eiter, T., Gottlob, G., and Voronkov, A. (2001). Complexity and expressive power of logic programming. ACM Computing Surveys 33(3): 374-425.

Decker, S., Erdmann, M., Fensel, D., and Studer, R., (1998). Ontobroker in a Nutshell, in Research and Advanced Technologies for Digital Libraries. 1998, Springer Verlag LNCS 1513.

Denecker, M., Kakas, A. C. (2002). Abduction in logic programming, Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part I (A. C. Kakas, and F. Sadri, eds.), Springer Verlag, pp. 402-436.

Duschka, O., Genesereth, M. (1997). Answering Recursive Queries Using Views. PODS 1997: 109-116

Firat, A., Madnick, S., and Siegel, M. (2000). The Caméléon Web Wrapper Engine, Proceedings of the VLDB2000 Workshop on Technologies for E-Services, pages 1-9.

Firat, A., Madnick, S., and Grosof, B. (2002). Knowledge Integration to Overcome Ontological Heterogeneity: Challenges from Financial Information Systems, Proceedings of the International Conference on Information Systems.

Firat, A., Madnick, S., Grosof, B., Financial Information Integration in the Presence of Equational Ontological Conflicts, 12th Workshop on Information Technology and Systems, Barcelona, Spain, 2002.

Firat, A., Madnick, S., Siegel, M.,The Cameleon Approach to the Interoperability of Web Sources and Traditional Relational DataBases, Proceedings of the 10th Annual Workshop On Information Technologies and Systems, Brisbane, Queensland, Australia, 2000.

Firat, A., Peleshchuk, D., Rao, P. (1999) IWrap: Instant Wrapper Generator, unpublished manuscript.

Firat, A., Zhu, H., Lee., P. (2003). Caméléon User Manual, unpublished manuscript.

Frühwirth, T. (1998). Theory and Practice of Constraint Handling Rules, Special Issue on Constraint Logic Programming (P. Stuckey and K. Marriot, Eds.), Journal of Logic Programming, Vol 37(1-3), pp 95-138, October.

Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, V., Ullman, J., Widom, J. (1995). Integrating and Accessing Heterogeneous Information Sources in TSIMMIS. In Proceedings of the AAAI Symposium on Information Gathering, pp. 61-64, Stanford, California, March 1995.

Gruser, J., Raschid, L., Vidal, M., Bright, L. (1998) Wrapper Generation
for Web Accessible Data Sources In Proceedings of CoopIS'98.

Ghidini, C., and Giunchiglia, F. (2001). Local Models Semantics, or Contextual Reasoning = Locality + Compatibility. Artificial Intelligence. 127(2):221-259.

Ghidini, C., and Serafini, L. (1998). Information Integration for Electronic Commerce. In Agent Mediated Electronic Commerce. First International Workshop on Agent Mediated Electronic Trading, AMET-98, Volume 1571 of LNAI. Springer.

Goh, C. H. (1997). Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Systems, MIT Ph.D. Thesis.

Grosof, B. (2001). "Representing E-Business Rules for the Semantic Web: Situated Courteous Logic Programs in RuleML", Proc. Workshop on Information Technologies and Systems (WITS '01), 2001.

Gruber, T. R. (1993). A Translation Approach to Portable Ontology Specifications, Knowledge Acquisition, 5, 199–220.

Guarino, N., Giaretta, P. (1995). Ontologies and knowledge bases: Towards a terminological clarification. In N.J.I. Mars, editor, Towards Very Large Knowledge Bases, IOS Press, 1995.

Guarino, N. (1995). Formal Ontology, Conceptual Analysis and Knowledge Representation, International Journal of Human-Computer Studies, 43, 625–640.

Guarino, N (ed.) (1998). Formal Ontology in Information Systems, Amsterdam, Berlin, Oxford: IOS Press. Tokyo, Washington, DC: IOS Press (Frontiers in Artificial Intelligence and Applications), 1998.

Guarino, N. and Welty, C. (2000). A Formal Ontology of Properties, in R. Dieng and O. Corby (eds.), Knowledge Engineering and Knowledge Management: Methods, Models and Tools. 12th International Conference (EKAW 2000), Berlin/New York: Springer: 97–112.

Guha R. V. (1991). Contexts: a formalization and some applications, MCC Tech Rep ACT-CYC42391.

Halevy, A. (2000). Theory of Answering Queries Using Views. ACM SIGMOD Record 29(4): 40-47.

Hammer, J., Garcia-Molina, H., Cho, J., Aranha, R., Crespo, A. (1997) "Extracting Semistructured Information from the Web". In Proceedings of the Workshop on Management of Semistructured Data. Tucson, Arizona.

Hirst, G. (2000). Context as a spurious concept. Proceedings, Conference on Intelligent Text Processing and Computational Linguistics, Mexico City, 273--287.

Horrocks, I., Fensel, D., Broekstra, J., Decker, S., Erdmann, M., Goble, C., van Harmelen, F., Klein, M., Staab, S., Studer, R., Motta, E. (2000). OIL: The Ontology Inference Layer. Technical Report IR-479, Vrije Universiteit Amsterdam, Faculty of Sciences.

Hsu, C., and Dung, M. (1998). Wrapping semistructured web pages with finite-state transducers. To appear in the Proceedings of the Conference on Autonomous Learning and Discovery CONALD-98.

Huck, G., Fankhauser, P., Aberer, K., Neuhold, E. (1998): JEDI: Extracting and

Synthesizing Information from the Web; submitted to COOPIS 98, New York, August, 1998; IEEE Computer Society Press.

Jakóbisiak, M (1996). Programming the Web- Design and Implementation of a Multidatabase

Browser, CISL WP#96-04, May 1996, MIT Sloan School of Management.

Kakas, A. C., Michael, A., and Mourlas, C. (2000). ACLP: Abductive Constraint Logic Programming, Journal of Logic Programming, 44(1-3):129-177.

Kakas, A. C., Michael, A. (1995). Integrating abductive and constraint logic programming. n To appear in Proc. International Logic Programming Conference.

Kaleem, M. B. (2003). CLAMP: Application Merging in the ECOIN Context Mediation System Using the Context-Linking Approach. Master of Engineering Thesis. Massachusetts Institute of Technology.

Kashyap, V., Sheth, A. P. (1996). Semantic and Schematic Similarities Between Database Objects: A Context-based Approach, The VLDB Journal, 5(4):276-304.

Kim, W., Seo, J. (1991). Classifying Schematic and Data Heterogeneity in Multidatabase Systems. IEEE Computer 24(12): 12-18.

Klyne, G. (2000). Contexts for RDF Information Modelling, http://public.research.mimesweeper.com/RDF/RDFContexts.html.

Knoblock, C., Lerman, K., Minton, S., Muslea, I. (2000) IEEE Data Engineering Bulletin, 23(4):33--41, December 2000.

Kowalski, R. (1974). Predicate Logic as Programming Language. In J. L. Rosenfeld, editor, Information Processing 74. Proceedings of IFIP Congress 74, Stockholm, pages 569--574, Amsterdam, August 1974. North-Holland.

Kowalski, R. A., (1992). A dual form of logic programming. Lecture Notes, Workshop in Honour of Jack Minker, University of Maryland, November.

Kuhn, E., Puntigam, F., Elmagarmid A. (1991). Multidatabase Transaction and Query Processing in Logic, Database Transaction Models for Advanced Applications, Morgan Kaufmann Publishers.

Kuhn, E., Ludwig, T. (1988). VIP-MDBS: a logic multidatabase system, Proceedings of the first international symposium on Databases in parallel and distributed systems, p.190-201, December 05-07, Austin, Texas, United States.

Kushmerick, N., Doorenbos, R., Weld., D. (1997) Wrapper Induction for Information Extraction. IJCAI-97, August 1997.

Lacroix, Z. (1999). Object Views through Search Views of Web datasources, International Conference on Conceptual Modeling (ER'99), Paris, France, November.

Laender, A., Ribeiro-Neto, B., Silva, A. and Teixeira, J. (2002). A Brief Survey of Web Data Extraction Tools, SIGMOD Record, Volume 31, Number 2.

Landers, T., Rosenberg, R. (1982). An Overview of MULTIBASE, International Symposium on Distributed Data Bases, 153-184

Lee, J. (1996). Integrating Information from Disparate Contexts: A Theory of Semantic Interoperability, Ph.D. Thesis, MIT.

Lee, P. W. (2003). Metadata Representation and Management for Context Mediation. Master thesis, Massachusetts Institute of Technology, Sloan School of Management, May.

Lenat, D., R. V. Guha, K. Pittman, D. Pratt, and M. Shepherd. (1990). Cyc: Towards programs with common sense. Communications of the ACM 33(8).

Lenzerini, M. (2001). Data Integration Is Harder than You Thought. CoopIS 2001: 22-26

Levy, A. (1998). "The Information Manifold Approach to Data Integration," IEEE Intelligent Systems, 1312-16.

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

Litwin, W. (1992). O*SQL: A language for object oriented multidatabase interoperability. In Proceedings of the Conference on IFIP WG2.6 Database Semantics and Interoperable Database Systems (DE-5) (Lorne, Victoria, Australia), D. K. Hsiao, E. J. Neuhold, and R. Sacks-Davis, Eds. North-Holland Publishing Co., Amsterdam, The Netherlands, 119-138.

Litwin, W Vigier, P. (1986) Dynamic Attributes in the Multidatabase System MRDSM. ICDE 1986: 103-110

Liu, L., Pu, C., Han, W., Buttler, D., Tang, W. (1999). XWrap: An Extensible Wrapper Construction System for Internet Information Sources, Oregon Graduate Institute of Science and Technology.

Lloyd, J. (1984). Foundations of Logic Programming, 1st Edition. Springer.

Luke, S., Spector, L., Rager, D., and Hendler, J. (1997). Ontology-based Web Agents. In Proceedings of the First International Conference on Autonomous Agents, 59-66. New York, NY: Association of Computing Machinery.

MacGregor, R., Chalupsky, H., Moriarty, D. and Valente, A. (1999). Ontology Merging with OntoMorph.

http://reliant.teknowledge.com/HPKB/meetings/meet040799/Chalupsky/index.htm

Manola, F. (2002). The Semantic Web and the Role of Information Systems Research, NSF-OntoWeb Invitational Workshop onDB-IS Research for Semantic Web and Enterprises.

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

McCarthy J., (1993). Notes on Formalizing Context, Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence.

McCarthy, John and Buvac, S, 1997. Formalizing context (expanded notes). In: Aliseda, A., van Glabbeek, R. and Westerstrahl, D., Editors, 1997. Computing natural language, Center for the Study of Language and Information, Stanford, CA.

McGuinness, D.L., Fikes, R., Rice, J. and Wilder, S. (2000). An Environment for Merging and Testing Large Ontologies. In: Proceedings of the Seventh International

Conference on Principles of Knowledge Representation and Reasoning (KR2000), Breckenridge, Colorado.

McGuinness, D., van Harmelen , F. (2003). OWL Web Ontology Language. http://www.w3.org/TR/owl-features/

Mecca, G., Merialdo, P., Atzeni, P. (1999). ARANEUS in the Era of XML - IEEE Data Engineering Bullettin, Special Issue on XML, September.

Miller, G. (1995). “WordNet: a lexical database for English”, Comm. of the ACM, 38(11):39–41.

Muslea, I., Minton, S., and Knoblock, C. (1998) STALKER: Learning extraction rules for semistructure, Web-based information sources.In Proc. of AAAI’98: Workshop on AI and Information Integration.

Niles, I & Pease A. (2001). Towards A Standard Upper Ontology. In Proceedings of FOIS 2001, , Ogunquit, Maine, USA.

Noy, N., F., Musen, M., A. (2000). PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. Proceedings of the National Conference on Artificial Intelligence 2000: 450-455

Peirce, C (1903). "Review of What is Meaning? by Victoria Welby and The Principles of Mathematics by Bertrand Russell," reprinted in Ketner & Cook (1979) pp. 143-145

Pottinger, R., Halevy, A., (2001). MiniCon: A scalable algorithm for answering queries using views. VLDB Journal 10(2-3): 182-198.

Qu, J. (1996). Data Wrapping on the Worl Wide Web, CISL WP#96-05, February 1996, MIT Sloan School of Management.

Reed, S. and Lenat, D. (2002). Mapping Ontologies into Cyc. AAAI-2002 Workshop on Ontologies and the Semantic Web, http://reliant.teknowledge.com/AAAI-2002/Reed.pdf

Alatovic, T. (2002) Capabilities Aware Planner/Optimizer/Executioner for COIN Project, Master of Engineering Thesis, Massachusetts Institute of Technology, 2002.

Roth, M.T., Schwarz, P.M. (1997). Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. VLDB'97, Proc. of 23rd Int. Conference on Very Large Data Bases, August 25-29.

Scheuermann, P., Elmagarmid, A. K., Garcia-Molina, H., Manola, F., McLeod, D., Rosenthal, A., Templeton, M. (1990). Report on the Workshop on Heterogenous Database Systems held at Northwestern University, Evanston, Illinois, December 11-13, 1989, Sponsored by NSF. SIGMOD Record 19(4): 23-31.Sciore, E., Siegel, M., Rosenthal, A. (1994). Using Semantic Values to Facilitate Interoperability Among Heterogeneous Information Systems, ACM Transactions on Database Systems, 19(2):254290.

Serafini, L., and Ghidini, C. (2000). A Context Based Semantics for Federated Databases. In M. Cavalanti, P. Bonzon, and R. Nossum (eds.), Formal Aspects of Context, volume 20 of Applied Logic Series. Kluwer Academic Publishers.

Sahuguet, A., Azavant, F. (1998). W4F: the WysiWyg Web Wrapper Factory. Technical

report, University of Pennsylvania, Department of Computer and Information Science.Siegel, M., Madnick, S. (1991). A Metadata Approach to Resolving Semantic Conflicts, Proceedings of the Seventeenth International Conference on Very Large Databases, pp. 133-145.

Siegel, M., Madnick, S. (2002). Seizing the Opportunity: Exploiting the Web Aggregation, MIS Quarterly Executive Vol. 1 No. 1 / March 2002.

Silberschatz, A., Zdonik, S., (1997). Database systems -- breaking out of the box. SIGMOD Record, 26, pp 36--50.

Smith, B., and Chris, W. (2001). Ontology: Towards a new synthesis. In Chris Welty and Barry Smith, eds., Formal Ontology in Information Systems. Pp. iii-x. Ongunquit, Maine: ACM Press.

Sowa, J. (1997). Peircean foundations for a theory of context, Proceedings of the International Conference on Conceptual Structures, 41-64.

Sperber, D., Wilson, D. (1986), Relevance: Communication and Cognition, Oxford: Basil Blackwell.

Spyns P., Meersman R., & Jarrar M. (2002). Data modelling versus Ontological engineering, SIGMOD Record Special Issue on Semantic Web, Database Management and Information Systems, December 31 (4).

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

Subrahmanian, V.S., Adali, S., Brink, A., Lu, J. J., Rajput, A., Rogers, T. J., Ross, R., Ward, C. (2000). HERMES A Heterogeneous Reasoning and Mediator System, http://www.cs.umd.edu//projects/hermes/overview/paper/index.html.

Tatbul, N., Karpenko, O., Convey, C. (2001). Data Integration Services, Technical Report, Brown University, Computer Science.

Tomasic, A., Raschid, L., and Valduriez, P. (1998). Scaling access to heterogeneous data sources with disco, IEEE Transactions on Knowledge and Data Engineering, 10(5): 808-823.

Ullman, J. (1991). Principles of Database and Knowledge-base Systems, Volumes I & II. Computer Science Press.

Ushold, M., Gruninger, M. (1996). Ontologies: principles, methods and applications; The Knowledge Engineering Review 11 2, pp 93-136.

Wache, H., Vogele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., Hubner, S. (2001). Ontology-based integration of information - a survey of existing approaches. In Stuckenschmidt, H., ed., IJCAI-01 Workshop: Ontologies and Information Sharing, 108--117.

Wiederhold, G, (1992a). The Roles of Artificial Intelligence in Information Systems, Journal of Intelligent Information Systems, 11(1):35-56.

Wiederhold, G., (1992b). Mediators in the Architecture of Future Information Systems, IEEE Computer, 25(3):38-49.