

8

# Derivation of Context Axioms and Semantics of Data with the Domain Model Editor

by

Christopher Leung

Submitted to the Department of Electrical Engineering  
and Computer Science  
in partial fulfillment of the requirements for the degrees of

**Master of Engineering in Electrical Engineering and Computer Science**  
and

**Bachelor of Science in Electrical Science and Engineering**

at the

**MASSACHUSETTS INSTITUTE OF TECHNOLOGY**

June 1998

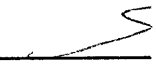
© Christopher C. Leung. MCMXCVIII. All rights reserved.

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


Signature of Author

\_\_\_\_\_  
Department of Electrical Engineering and Computer Science  
May 20, 1998

Certified by

  
\_\_\_\_\_  
Stuart E. Madnick  
John Norris Maguire Professor Of Information Technology  
Thesis Supervisor

Accepted by

  
\_\_\_\_\_  
Arthur Smith  
Chair  
Department Committee on Graduate Students

Eng.

# **Derivation of Context Axioms and Semantics of Data with the Domain Model Editor**

by

Christopher C. Leung

Submitted to the Department of Electrical Engineering and Computer Science  
in partial fulfillment of the requirements for the degrees of

**Master of Engineering in Electrical Engineering and Computer Science**  
and

**Bachelor of Science in Electrical Science and Engineering**

## **Abstract**

The Context Interchange Strategy [Siegel and Madnick, 1991] presents a novel perspective for mediated data access in which the Context Mediator detects and reconciles semantic conflicts among heterogeneous systems through comparison of contexts associated with any two systems such as a database and a web site. Examples of conflicts are use of different currencies and scale factors for representing the money amounts. To resolve these conflicts, the Context Interchange System generates mediated queries that take into account all the possible conflicts when retrieving the data from the sources. This involves the Context Mediation process [Goh, 1996], which is made possible by representing semantics of the data source in the form of contexts and domain models, which will be interpreted by the context mediator and transformed into a mediated query. A User can define and hand-code the context of a data source, but it is difficult to visualize and understand how each semantic type relates to the others in text format. It also requires users a lot of time to learn how to do the coding.

To solve this problem, I designed a system, which consists of: 1) the domain model editor, a graphical tool which enables the users to define the domain model graphically using a web browser; 2) the domain-to-context translator, which generates the domain model into source code written in PENNY; and 3) the remote method invocation tool, which stores the data source information in the host machine obtained from the client, and allows the PENNY compiler to generate the information to be evaluated by the Context Mediator.

Thesis Supervisor: Stuart Madnick

Title: John Norris Maguire Professor of Information Technology



# Acknowledgments

As part of the work in the Context Interchange Project, this thesis constitutes years of research and design work by the Context Interchange Group. It brings to me an precious and exciting experience as I enter the next phase of my life.

I would like to acknowledge and thank all the people in the group who have contributed their time and effort in the world of Information Technology. First of all, I would like to thank Professor Stuart Madnick for his admired leadership and inspirational research work that started the Context Interchange Project. I would like to thank Dr. Michael Siegel for giving me the opportunity to work for this project. His generosity and constructive advice create the key driving forces that make me work harder and do better. I would also like to express my gratitude to Dr. Stephane Bressan, Tito Pena, Allen Moulton, and Natalia Levina who have provided invaluable support to me in carrying out the design work of my thesis project, as well as to my coworkers Ahmed Shah, Steven Tu, Ricardo Ambrosia who have taken their precious time sharing ideas with me regarding the project. And my appreciation to my friends (in no particular order) Priscilla Lee, Lawrence Yee, Jervis Lui, Henry Tang, Ernie Yeh, Ed Kim, Wai Kit Lau Mike Shao, James Chen, Yu Zhou, Preston Li, Albert Wong and Francis Wong, who have shared friendship and a fun and memorable college life with me. Thanks for their kindness and genuine concern when I was in grief.

I would like to express my deep love to my family, especially my parents Michael and Mary, who have granted me unconditional love throughout my life; my brother Terence, my sister Gladys, who have given me all kinds of support in both my career and my life; and my beloved nanny Man Ying Wong, who has spent a lot her effort in bringing me up and helping my entire family. Finally, I would like to thank God, for His love, mercy, and forgiveness. Without Him, nothing can happen.

# Contents

- 1. Introduction to Context Interchange** 6

---
- 1.1 Motivation of the Project..... 7
- 1.2 Thesis Outline ..... 9
  
- 2. Construction of the Context Interchange Data Model** 10

---
- 2.1 Overview of the Context Interchange System..... 10
- 2.2 Integration Scenario ..... 14
- 2.3 The COIN Framework..... 20
- 2.4 Domain Modeling in COIN..... 30
  
- 3. Concepts of Mapping Data to the Domain Model with the PENNY Language** 34

---
- 3.1 Defining Semantics Relations in the Domain Model..... 35
- 3.2 Mapping Sources to the Domain Model via Elevation Axioms..... 37
- 3.3 Defining Context with Context Axioms ..... 40
  
- 4. Design and Implementation of the Domain Model Editor** 43

---
- 4.1 Overview of the Context Interchange Prototype ..... 43
- 4.2 Design of the Domain Model Editor System ..... 45
- 4.3 Implementation of the Client Application with Java ..... 51
  
- 5. Conclusion: Limitations and Future Work** 54

---
- 5.1 Using Java Development Kit..... 54
- 5.2 Domain Model Editor..... 56
- 5.3 Integration with the Current COIN Components ..... 57
  
- Bibliography** 58

---

**Appendix A User Guide and Programmer’s Reference** 60

---

A1-1 Getting Started ..... 65

A1-2 Using the Graphical User Interface ..... 66

A1-3 Graphical Representation of the Domain Model ..... 68

A1-4 Using the Code Generation Wizard ..... 70

A1-5 Saving and Loading Different Formats of Documents ..... 77

A2-1 Class Hierarchy ..... 81

A2-2 Functional Specifications of Each Class and Description of All Functions ..... 83

# List of Figures

Figure 2-1 Illustration of Data Retrieval Problems from Multiple Databases .....	12
Figure 2-2 Architecture of the Context Interchange System.....	13
Figure 2-3 Looking for Company's Financial Information from Disclosure.....	16
Figure 2-4 A Subset of Financial Data retrieved from Disclosure by the Wrapper on 05/10/98 .....	16
Figure 2-5 Subset of Data from Worldscope compared to Data from Disclosure .....	18
Figure 2-6 Subset of Currency Data from Olsen web site .....	19
Figure 2-7 A summary of how queries are processed within the Context Interchange strategy .....	22
Figure 2-8 Context Mediator Internals.....	24
Figure 2-9 Assumptions of Sources and User's Preference.....	27
Figure 2-10 Queried Results from Worldscope without Mediation .....	28
Figure 2-11 Mediated Queried Results using Exchange Rates from Olsen.....	30
Figure 2-12 Graphical Representation of Different Components in the Domain Model....	31
Figure 3-1 The Domain Model for the Integration Scenario .....	35
Figure 4-1 Architecture of the Context Interchange (COIN) System .....	44
Figure 4-2 The Architecture of the Domain Model Editor System.....	46
Figure 4-3 Components of the Domain Model Editor .....	47
Figure A-1 The Domain Model Editor .....	66
Figure A-2 The File and Edit Menus .....	67
Figure A-3 The View Menu.....	67
Figure A-4 Graphical Representation of Different Components in the Domain Model.....	69
Figure A-5 Creating new system and semantic nodes and changing the names.....	70
Figure A-6 The Code Generation Wizard Box .....	71
Figure A-7 Generating Context Axioms.....	73
Figure A-8 Assigning Entities to the Domain Model .....	74
Figure A-9 Generating Elevation Axioms .....	75
Figure A-10 Generating Elevation Axioms .....	76

# Chapter 1

## Introduction to Context Interchange

In the fast-growing Information Technology world, database systems are essential tools for companies and organizations to store large amount of data related to their businesses across different parts of the world. With the introduction of the Internet, the World Wide Web provides an easy way to quickly access data from different database systems. Many researchers have been paying close attention to the various issues regarding database system integration because the number of information sources that are being physically connected has grown rapidly. With this growing abundance of data existing in different systems, the problem of understanding and interpreting it all has become a definite challenge [Madnick, 1996]. While the Internet has provided an excellent infrastructure for physical connectivity (the ability to exchange data) among disparate data sources and receivers, it has problems providing a logical connectivity (the ability to exchange data meaningfully) among them [Pena, 1997]. The real problem occurs when the data sources and the receivers maintain different assumptions about the data being exchanged [Goh, 1996]. We refer to such difference as the *contextual difference* among sources.

One example that illustrates this problem is the handling of dates such as the beginning date of the fiscal year. The date “7/1/97” means July 1, 1997 in American standard whereas it means January 7, 1997 to a European. Another example would be the use of the dollar sign “\$.” It can represent amount of money in U.S. dollars, but it can also represent Hong Kong dollars. The data would have a different meaning, depending upon

different assumption about the data. Under these circumstances, *physical connectivity* does not lead to *logical connectivity*. Such issue is referred to as the *semantic interoperability among autonomous and heterogeneous systems* [Sheth and Larson, 1990].

The Context Interchange (COIN) provides a strategy that gives a novel approach to detecting and reconciling semantic conflicts [Siegel and Madnick, 1991] between any two systems. The logical connectivity of two systems can be essential when manipulating the information because the meaning of this information can depend on a particular *context* of the data source. In essence, each information source is associated with a context [Pena, 1997], which is a set of axioms describing certain assumptions about the data. An attempt by the Context Interchange to identify a formal conceptual basis of systems during the Context Mediation process is the use of the *domain model*, a derivation of the *semantic-value model* [Sciore, Siegel, and Rosenthal, 1994] that defines the data property of a context. The provision of the *Context Mediation* service by the Context Interchange [Siegel and Madnick, 1991] only requires that the user furnish a logical (declarative) specification of *how data are interpreted* in the form of a domain model and *how conflicts should be resolved*. The *Context Mediator* that provides such service will take care of detecting and reconciling semantic conflicts, and these operations are designed to be transparent to the user.

## 1.1 Motivation of the Project

In order for the Context Mediator to identify and reconcile the conflicts that may arrive from evaluating the data from different data sources, we need to define a data model that specifies how the data are interpreted in terms of the contexts of their sources. In COIN, the model used is called the *Domain Model*. It is a “logical” data model that uses mathematical logic as a way for representing the *values* of data and a language for expressing operations on an underlying data structure. It is also an “object-oriented” data model that supports object-identity, type hierarchy, inheritance, and overriding methods. The main purposes of deriving the Domain Model for Context Mediation are to:

- Describe the semantic objects and relations presented in the context of the sources, e.g. databases and web sites;
- Map the data through the definition of Elevation Axioms; and
- Define methods for converting the data from one context to another with the definition of Context Axioms.

As the Context Interchange Project began in 1996, the Domain Model is defined in a language called COINL [Goh, 1996]. The expressions and symbols used in this language are meant to illustrate the concepts of defining semantics relationships and data mapping in the Domain Model. Its complexity makes it hard to read and inconvenient to program because the various symbols do not appear on a keyboard. About a year later, the PENNY language and the PENNY compiler [Pena, 1997] were implemented to simplify COINL by redefining the expressions in a more readable format<sup>1</sup>. However, defining the new language is not the complete solution because one would still have a hard time visualizing and understanding the complex relationships between the data in the Domain Model. In addition, as we shall discuss in Chapter 4, the source code resides in the server system and is hard to be retrieved by an end user who wants to read or edit the code. In order to resolve these problems, I designed the Domain Model Editor System which:

- Helps the users, who have trouble understanding or deriving the Domain Model in COINL or PENNY, better visualize the relationships between data with a graphical representation of the Domain Model; and
- Provides a user friendly programming environment for users to derive the Domain Model, map the data from the sources to the model, and define conversion functions for the Context Mediation process.

---

<sup>1</sup> For more details including the syntax of PENNY, please refer to Chapter 4.

## 1.2 Thesis Outline

The previous sections briefly explain the use of Domain Model as a basis for reconciliation of data in COIN. This thesis will give more insightful details on how the construction of the model helps the process of Context Mediation.

**Chapter 2, “Construction of the Context Interchange Data Model,”** gives an overview of the COIN framework and the data modeling concepts behind it. It describes how the Context Mediator interprets the Domain Model and generates mediated queries in order to return appropriate answers to the users. The chapter includes an example that explains a data integration scenario that involves data modeling and mapping in three financial data sources, namely Disclosure, Worldscope, and Olsen.

**Chapter 3, “Concepts of Mapping Data to the Domain Model with PENNY Language,”** goes into details on how to actually define the Domain Model in the PENNY language. It also explains how to map the data sources to the Domain Model via Elevation Axioms represented in PENNY; and how to define the conversion functions with the Context Axioms.

**Chapter 4, “Design and Implementation of the Domain Model Editor,”** looks into the system perspective of Context Interchange, particularly the Domain Model Editor. It discusses the advantages and disadvantages of implementing the Editor in Java and Java RMI based on different design issues such as extensibility, portability, security, etc.

For conclusion, **Chapter 5, “Conclusion: Limitations and Future Work,”** discusses the limitations in the current versions of the Java Development Kit and of the Domain Model Editor, and the possible future work that might improve the current implementation.

At last, **Appendix A, “User’s Guide and Programmer’s Reference,”** provides a step-by-step guide on how to run and use the Domain Model Editor. It is also a complete Programmer’s Reference for anyone who is interested in extending this application.



## Chapter 2

# Construction of the Context Interchange Data Model

The focus of this chapter is to describe the composition of the Context Interchange data model that is used by the Context Mediator to answer an SQL query submitted by the user. The first section gives an overview of the components in the COIN System that are responsible for evaluating the Domain Model and generating *mediated queries* for answering users' queries. As a practical example, the second section presents an integration scenario that introduces some conflicts which might appear when retrieving data from different data sources. It also describes how the Context Mediation process can reconcile these conflicts by identifying the contextual differences from the context definition of each source. The last two sections will go further into explaining how to construct the Domain Model as the knowledge base of contexts for Context Mediation, and how to map to the Domain Model.

### 2.1 Overview of the Context Interchange Architecture

Context Interchange, or COIN, provides a mediator-based architecture for logical connectivity among disparate data sources. A conventional database query only retrieves the “raw” data directly from a database. It does not take into account the real meanings of

such data as well as the conflicts that might arrive from integration among multiple data sources.

### 2.1.1 Major Operation in Context Interchange

The major goal of Context Mediation is to rewrite the user's SQL query such that the data exchange among sites can be achieved under consistent conditions, for example, by converting all money amounts to a single currency, keeping the same date format as "mm/dd/yy", and using a scale factor of 1, etc. Suppose that a user, for example, a financial analyst, wants to know which publicly-traded Japanese companies were profitable last year from database A (dbA). The user also wants to find out the stock prices of these companies at the end of last fiscal year from database B (dbB). The user can formulate an SQL query as follow:

```
SELECT dbA.companyName, dbA.revenue, dbB.stockprice FROM dbA, dbB
WHERE dbA.revenue > dbA.expense
AND dbA.companyName = dbB.tickername AND dbA.fiscalDate = dbB.date;
```

The query is trying to look for any company in database A which has positive profit (revenue > expense), match its name with the corresponding entry found in database B, and list its the stock price on last day of the fiscal year. The results will contain the name, revenue and stock price of any company with more than zero profit. For the inquired results in this example to display the correct values, the following assumptions have to be true:

- **Use of Names**

In order to retrieve the company's information from both databases, the convention for company names must be consistent. Generally, a stock price stored in a database is referred by the ticker name of the company instead of its official name. For instance, database B may use "HMC" to represent the name for Honda Motor CO. Therefore the above query entered by the user will not be able to match the right names in both databases. This problem is presented in Figure 2-1 as an illustrative example.

- **Date Format**

The fiscal date in database A and the date that the stock price corresponds to in database B must be in same format (both in “mm/dd/yy” or “dd/mm/yy”). Otherwise, the stock price might be obtained from the wrong day or it might not be found. Figure 2-1 shows that database A uses “mm/dd/yy”, whereas database B uses “dd/mm/yy.”

Database A — Listed: Company Name, Revenue, and Expense

cname	revenue	expense	fiscaldate
Honda Motor Co	5,290,000,000	5,080,000,000	31/03/98
Toyota Motor Corp	7,200,000,000	6,890,000,000	31/03/98
...	...	...	...

Database B — Listed: Company’s Ticker Name, Date of Trade

tickername	stockprice <sup>2</sup>	date
HMC	63.875	03/31/98
TOYOY	50.625	03/31/98

Figure 2-1 Illustration of Data Retrieval Problems from Multiple Databases

- **Currency**

As shown in Figure 2-1, database A might hold money amounts in Japanese Yen (¥) while database B might present the stock prices in US Dollars (\$). The users would be confused by the mixture of different currencies and could make mistakes as they do calculations based on these two values.

The Context Mediator helps reconcile the conflicts like those described above by taking into account all the contextual differences among the data sources. There may be more conflicts arriving from data integration, as described in Section 2.2. Section 2.3, “Query answering by the Context Mediator” will discuss the mediation process in more details.

## 2.1.2 Context Interchange Architecture

Let us first look at the overall architecture of Context Interchange System and the responsibilities that each component takes:

---

<sup>2</sup> Conventionally, the stock prices are presented in fractions, e.g. 63.875 should have been 63 7/8 as seen in web sites like: <http://quotes.yahoo.com/>. This example is showing decimals for simplicity.

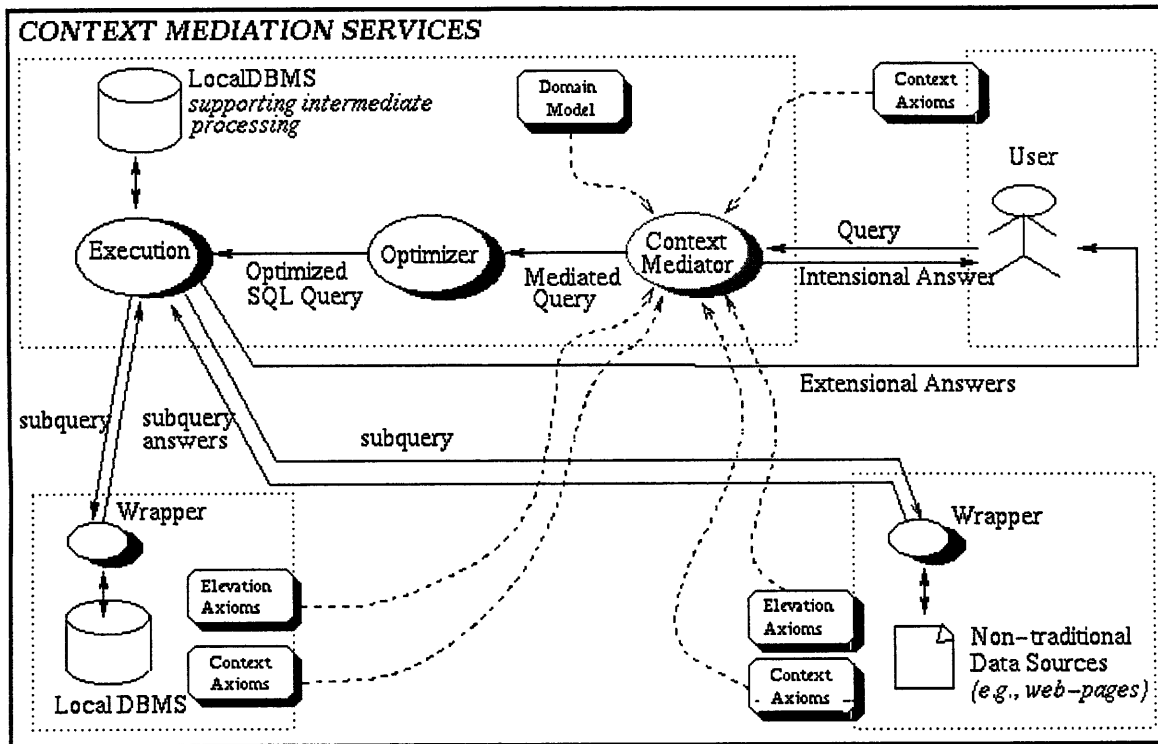


Figure 2-2 Architecture of the Context Interchange System

The primary operations of Context Interchange include:

- **Context Mediator**

The Context Mediator, as seen in Figure 2-2, translates an SQL query submitted by an *end-user* to a well-formed **mediated query**. The translation is based on the semantics information provided from the Domain Model, as well as the Context and Elevation Axioms that describe the source by specifying the semantic relationships and conversion functions. An example of an end-user can be a financial analyst who wants to look at some financial information that may be derived from the data provided by several vendors' database servers and web sites. Section 2.3 will describe more of this process.

- **Optimizer**

Generates an optimized SQL Query plan that reduces some redundancy in the mediated query. This process helps to reduce the number of possible query statements and hence speeds up data retrieval from the sources.

- **Executioner**

Executes the plan by dispatching subqueries to obtain data from the sources, collates and operates on the intermediary results, and returning the final answer to the user who submits the query. There are two processes that serve as data collection tools for the outside sources: 1) *database gateways*, which provide physical connectivity to databases and network and 2) *wrappers*, which collects the data from the web pages generated by the web sites. The goal is to insulate the Mediator Processes from the idiosyncrasies of different database management systems by providing a uniform protocol for database access as well as a canonical query language for formulating the queries [Goh, 1996].

Chapter 4 will give a more detailed system perspective view of the Context Interchange System. This chapter will focus more on the composition of the COIN data model necessary for the Context Mediation process, as described in the following sections.

## **2.2 Integration Scenario**

This section gives an example that helps the users understand how the data from different sources, namely Disclosure, Worldscope<sup>3</sup>, and Olsen, are mapped together and integrated in the Context Interchange system. It shows that every data source may have a different context associated with it, for example, it may use a different date format, or the money amounts may use different scale factors and currencies. Such conflicts would cause difficulties or even problems in deriving some values from the data of more than one sources. To resolve this problem, the Context Mediator in the Context Interchange System uses a strategy that maps the data to the Domain Model, which consists of 1) semantic types that represent these data, and 2) conversion functions that finds the output value based on both the context of the source and of user's preference.

---

<sup>3</sup> For more information on Disclosure and Worldscope, please visit their web sites at: <http://www.worldscope.com/> and <http://www.disclosure.com/>

First of all, let us look into the data available in the sources:

### **Disclosure**

A unit of Primark Corporation, Disclosure is a provider of financial information delivered via online and CD-ROM-based research tools. Subscribers of the Disclosure's web site have access to company financial and management information, updates and analysis of insider trading, and business news from over 2,500 publications and news wires. The company also provides data service for other web sites such as CNNfn (<http://cnnfn.com>) and Quicken (<http://www.quicken.com/>).

Disclosure uses the Oracle's relational database server<sup>4</sup> for storing the data. Generally, to retrieve data from a database, one has to write a procedure that sends SQL (Standard Query Language) queries to the server and returns column(s) of data as results; whereas in the Context Interchange System, the *wrapper* does the data gathering job by retrieving the data from a web page generated by the source — Disclosure in this case.

As an example, we look at Chrysler Corporation's information from the Disclosure web site:

---

<sup>4</sup> Oracle Corporation (<http://www.oracle.com/>) has released a new object-relational database server (Version 8) which combines object-oriented designing structures with current relational databases. Any Oracle Server whose version is 7.3.x or below is a relational database.

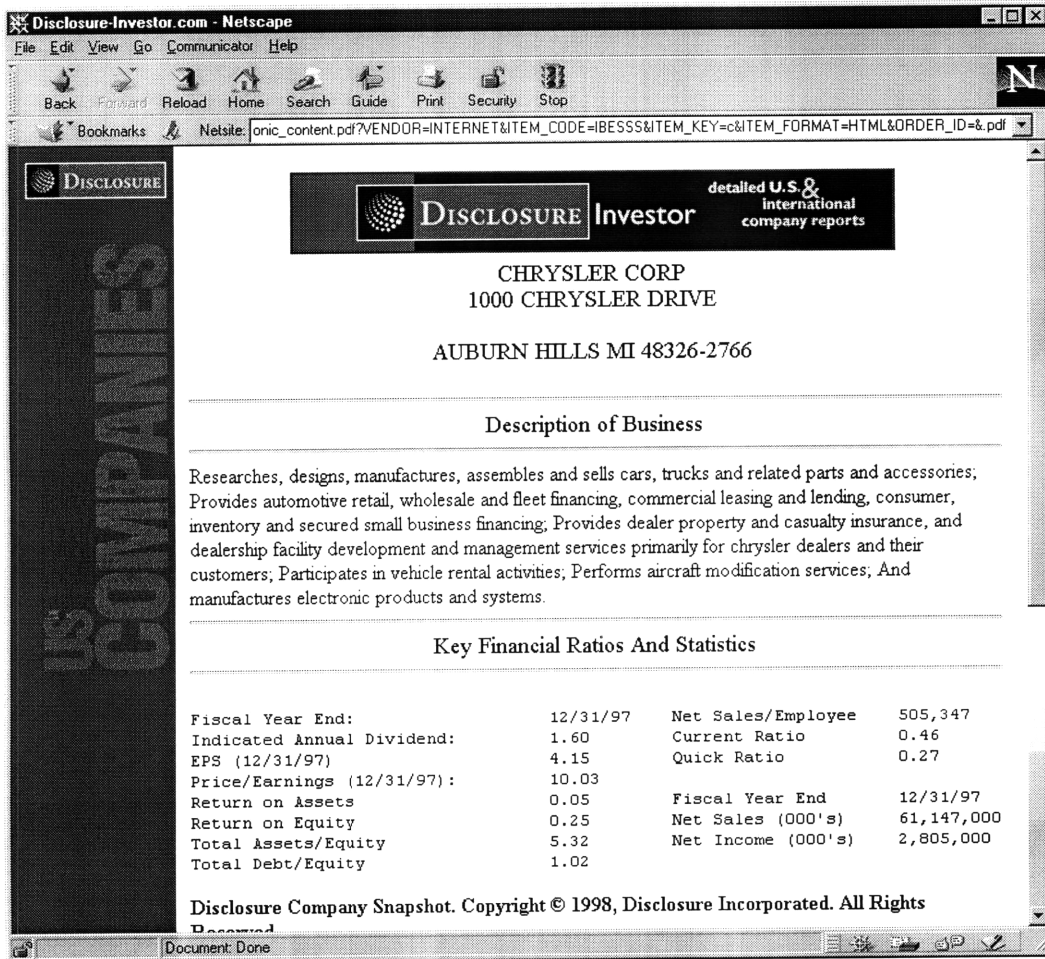


Figure 2-3 Looking for Company's Financial Information from Disclosure

We enter the name "Chrysler Corp" in the search form and the site returns the information shown in the above web page. It displays a handful of financial information useful for analyzing the performance of a company, but it may be very time-consuming for a user who wants to obtain similar information for 3 different companies because the user has to repeat the same search 3 times. The wrapper of the Context Interchange does a faster job by sending 3 queries to obtain the following list or portfolio of data:

Company Name	Fiscal Date	Net Income	Net Sales	Country
'CHRYSLER CORP'	12/31/97	2,805,000	61,147,000	'UNITED STATES'
'FORD MOTOR CO'	12/31/96	4,446,000	146,991,000	'UNITED STATES'
'HONDA MOTOR CO LTD'	03/31/97	221,168	5,293,302	'JAPAN'

Figure 2-4 A Subset of Financial Data retrieved from Disclosure by the Wrapper on 05/10/98

The Disclosure database consists of a single relation:

```
discaf(companyName, fiscalDate, netIncome, netSales, locationIncorp)
```

Each item inside the brackets represents the names of the column of data in the database table. The relation describes the financial performance of each company. There are a few notes regarding the “meanings” of the data of this site:

### 1. **Choice of Currency**

First, take a look at all the numbers in the above table. Without any prior knowledge of the source, one would be confused about the currency that the money amounts represent: Is the Honda’s net sales “5,293,302” in U.S. Dollars (\$) or Yen (¥), or in another currency? (According to the web site, all the values are in U.S. Dollars.)

### 2. **Use of Scale Factor**

Second, notice the labels “Net Sales (000’s)” and “Net Income (000’s)” inside the web page in Figure 2-1: the net income and net sales are presented in thousands of dollar units. The Context Mediator, which evaluates and integrates such values, must have known the existence of the scale factor. Otherwise, it would read the wrong net sales as “5,293,302” instead of “5,293,302,000.”

In the above example, we discover the “ambiguities” of data presented in a *single* data source. The introduction of another data source would possibly cause more *conflicts* due to the contextual differences in the way they present the data. Let us look into another web site as an example.

## **Worldscope**

Worldscope is an international financial database useful for portfolio managers, securities analysts, corporate finance professionals and researchers. It covers more than 90% of the world's market value, with 10 to 18 years of historical data on over 15,000 public companies in more than 50 developed and emerging markets. Worldscope also stores the data in Oracle database servers, but the relation is represented differently as:



worldAF(companyName, latestFinDate, netIncome, netSales, currency)

The following gives a table of data queried from Worldscope as a comparison to data from Disclosure:

WORLDSCOPE — worldAF(companyName, latestFinDate, netIncome, netSales, currency)

Company Name	Fiscal Date	Net Income	Net Sales	Currency
'CHRYSLER CORP'	12/31/97	2,804,900,000	61,146,900,000	'USD'
'FORD MTR CO'	12/31/96	4,450,000,000	147,000,000,000	'USD'
'HONDA MTR CO LTD'	03/31/97	29,393,227,200	588,479,835,800	'JPY'

DISCLOSURE — discsf(companyName, fiscalDate, netIncome, netSales, locationIncorp)

Company Name	Fiscal Date	Net Income	Net Sales	Country
'CHRYSLER CORP'	12/31/97	2,805,000	61,147,000	'UNITED STATES'
'FORD MOTOR CO'	12/31/96	4,446,000	146,991,000	'UNITED STATES'
'HONDA MOTOR CO LTD'	03/31/97	221,168	5,293,302	'JAPAN'

Figure 2-5 Subset of Data from Worldscope compared to Data from Disclosure

Let us take a look at the major differences between these two formats that might cause confusion to users who are looking at these two sites:

1. **Conflict over scale factor and currencies**

First notice the differences in the net income and net sales of say, HONDA MOTOR CO shown in each of the two tables. This is due to the use of different scale factors and currencies by each web site: For Worldscope, Honda's net income, represented as "29,393,227,200" is in Japanese Yen (¥) with a scale factor of one, whereas for Disclosure, its net income: "221,168" is in thousands of U.S. Dollars (\$). These two numbers are conceptually the same thing but represented in different *contexts*.

2. **Conflict over company name**

Observe the different conventions in naming the companies by the two web sites. Worldscope is using an abbreviated term 'MTR' in place of 'MOTOR' probably for saving memory space. Unlike conflicts such as using different scale factors (1 → 1000), the conversion in names are idiosyncratic and there are no systematic rules for mapping from one representation to another. One might refer to some multiple

widely-adopted standards for acquiring proper company names (e.g. Reuters, Dun & Bradstreet).

As described in the beginning of this chapter, we know that in order to ensure accurate results in evaluating the data, it is very important that the Context Mediator know about all the possible conflicts arriving from contextual differences. To achieve this, we have to derive the domain model, context axioms, elevation axioms, and conversion functions for the Context Mediator to reconcile these conflicts. The following sections will explain how to create the context information and formulate the conversion functions based on the integration scenario..

## Olsen

Before going on, we introduce another web site that provides the **exchange rates** and the **reference dates** for the Context Mediator in order to convert the money amount to the appropriate values. Figure 2-5 shows the data retrieved from Olsen for this example:

Olsen — exchange( sourceCurr, targetCurr, date, rate )

Source Currency	Target Currency	Date	Rate
'Japanese Yen'	'US Dollar'	31/12/96	.0086
'Japanese Yen'	'US Dollar'	31/12/97	.0092
'Japanese Yen'	'US Dollar'	31/03/97	.0090

Figure 2-6 Subset of Currency Data from Olsen web site

From the Olsen database, which has a relation of:

**exchange( fromCur, toCur, date, rate )**

we can retrieve the currency exchange rates valid for a particular date. There are a few considerations we should make notice of before using the data from such database and also databases in general:

### 1. Use of a different format in date

The dates in Figure 2-3 show that the format is in European style “dd/mm/yy” whereas from Figure 2-1 and 2-2 that date formats are in American style “mm/dd/yy.”

That means when we are mapping the date to the Domain Model, we must apply

conversion functions to change the format so that the Context Mediator will not get confused.

## 2. Whether use of such reference date is correct

Currency conversion requires the use of an appropriate reference time-point, which may be different from site to site. For example, one source may use the latest financial-reporting date as reference, while the other may use the exchange rate “as of” the day the money amount is loaded into the database. In general, this information constitutes a piece of meta-data which needs to be made explicit in the corresponding context if variations in change rates are important for the problem domain we are interested in.

## 2.3 The COIN Framework

According to Goh [1996], the COIN framework builds on the COIN data model to provide a formal characterization of the Context Interchange strategy for the integration of heterogeneous data sources. The COIN *framework*  $F_C$  is a quintuple  $\langle D, E, C, S, \mu \rangle$ . It consists of:

- a domain model,  $D$ , which present the definitions for the types of information units called semantic types. It constitutes a common vocabulary for capturing the semantics of data in disparate systems;
- the elevation set,  $E$ , a multi-set  $\{E_1, \dots, E_m\}$ , which is the set of elevation axioms  $E_i$  corresponding to  $s_i$  in the source set  $S$ . Each  $E_i$  creates a semantic relation corresponding to each extensional relation, and which defines the semantics of objects in this semantic relation by relating them to types and their attributes in the domain model;
- the context set,  $C$ , consisting of a collection of context providing a condensed definition of the semantic of data with no references to underlying schemas;

- the source set,  $S$ , is a multi-set with labels  $\{s_i := S_1, \dots, s_m := S_m\}$ . The label  $s_i$  is the name of the source.  $S$  is the collection of facts in the sources and the relevant integrity constraints of each; and
- source to context mapping,  $\mu$ , which identifies each data source with some context in the context set. The relation is defined by  $u(s_i) = c_j$ , where the source  $s_i$  is in context  $c_j$ .

One of the best features of this scheme is that it allows the semantics of data element to be described at different levels of specificity independent of how data is physically structured in the underlying system. For example, all the financial numbers such as *net income* and *net sales* reported in US Dollars in Disclosure can be mapped as instances of a *semantic object*, whose properties (currency = ‘USD’, scale factor = 1000, etc.) and modifier functions are pre-defined in the Domain Model. Thus, the system does not require administrator to explicitly define the semantics and modifier functions for every piece of data. One disadvantage is that the process of defining the Domain Model and context for each source can be complicated and time consuming. This is part of the motivation for implementing the graphical Domain Model Editor.

The COIN data model is a customized subset of the deductive object-oriented model called Gulog [Dobbie and Topor, 1995]. COIN can be described as a “logical” data model for representing knowledge and for expressing operations [Goh, 1996]. The COIN framework can be translated to a *normal program* [Lloyd, 1981] (equivalently, a *Datalog<sup>neg</sup>* program) that defines the semantics and computational procedures for query answering. Under this logical framework, the COIN presents no real distinction between factual statements (i.e., data in sources) and knowledge (i.e., statements encoding data semantics). Therefore, both queries on data sources (*data-level queries*) as well as queries on data semantics (*knowledge-level queries*) can be processed in an identical manner.

COIN can also be described as an “object-oriented” data model because it adopts an “object-centric” view of the world [Goh, 1996] and supports many of the features commonly associated with object-orientation, for example, object-identity, type hierarchy, inheritance, and overriding). In COIN, we use the Domain Model as a way to represent

this structure. It uses data abstraction to represent the “meanings” of data as *semantic objects*, which inherit from *primitive objects* (e.g. string, number); and to model conversion functions (for transforming the representation of data between contexts) as *methods* that are associated with these semantic objects. Section 2.4 will give more explanations.

We investigate on the adoption of an abductive framework [Kakas et al., 1993] as one possible way for query processing during Context Mediation. Under the abductive framework as we shall explain in Section 2.3.2, the Context Mediator compute the *intensional answers*, which can be interpreted as the corresponding *mediated query* in which database accesses are interleaved with data transformations required for mediating potential conflicts.

### 2.3.1 Overview of the COIN Query Framework

Figure 2-7 illustrates how the Context Interchange System evaluates the queries:

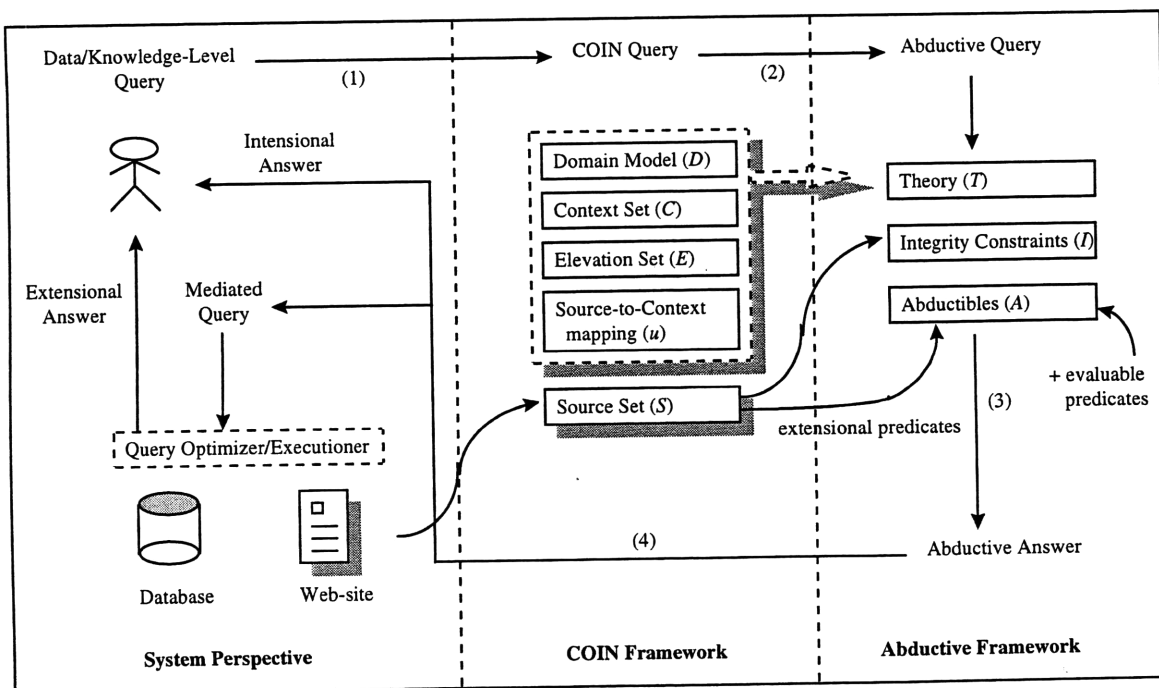


Figure 2-7 A summary of how queries are processed within the Context Interchange strategy

As we see in the above diagram, the COIN framework consists of three levels that process the queries entered by users and returns the answers that they want.

1. Transforms an SQL query to a well-formed COIN query;
2. Performs the COIN to DATALOG<sup>neg</sup> translation;
3. Abduction computation which generates an abductive answer corresponding to the given query; and
4. Transforms the answer from clausal form back to SQL.

The power of this system is that all of the intermediate processes are *transparent* to the users. This means that the users submitting the queries need not worry about the underlying assumptions about the data sources (such as use of currency or scale factor), because the system is already taking care of it. What matters is the output format that the users requested, and the system allows them to select a context for reporting the values of the output data. Although we are saying that the processes are transparent to the users, they can still find out the source information such as the context definitions and semantics relation from the Domain Model Editor that I implemented in this project.

### 2.3.2 Query Answering by the Context Mediator

This section briefly describes the query answering process done by the Context Mediator based on the *abductive framework*. To better illustrate the mediated results generated by the Context Mediator, we use the integration scenario discussed in Section 2.2.

The task of the Context Mediator is to use abductive reasoning to rewrite the SQL query submitted by the user. This is to make sure that all the possible data conflicts that may happen during the data integration are being taken care of. The resulting query, the *mediated query*, contains expressions that will provide an *intensional answer* for the original query. As seen in Figure 2-8 below, the Context Mediator consists of the following processes:

1. SQL-to-HC compilation — Compiles the user-defined SQL query and context (the output format that the user wants) into a Horn-clause (HC), or equivalently statements in Datalog<sup>neg</sup> used for computing the abductive answer;

2. COIN-to-HC translation — Transforms the COIN framework to the Horn-clause (HC) framework;
3. Abduction — Deduces the abductive answer from the output from 1 and 2; and
4. HC-to-SQL Compilation — Translates the answer in HC back to SQL format. The output of the whole process will be the mediated SQL query.

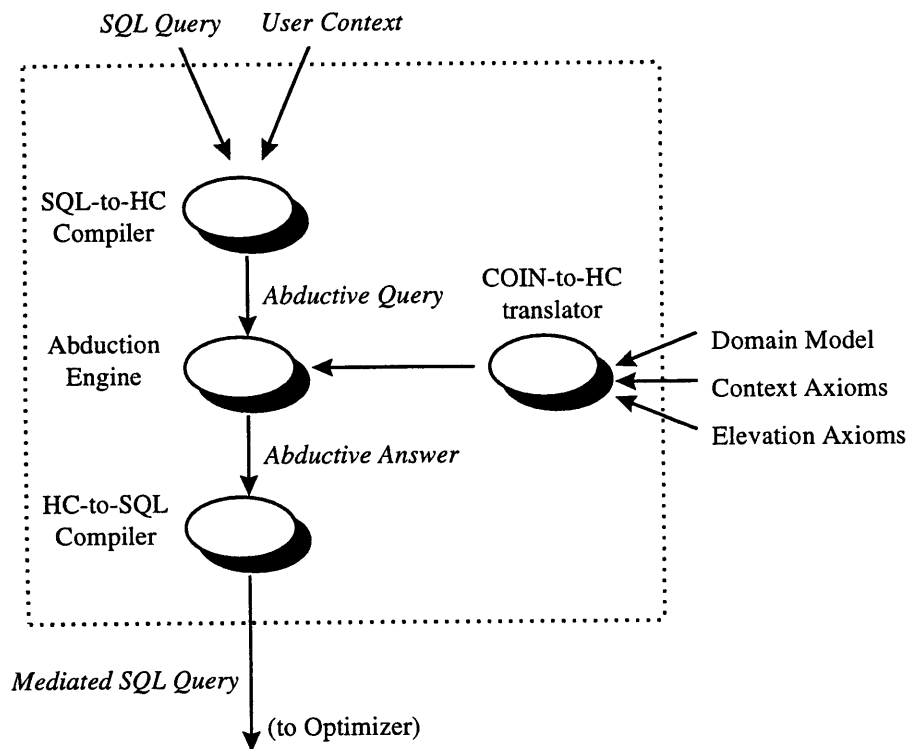


Figure 2-8 Context Mediator Internals

Following the Context Mediation process, the Optimizer (reduces the redundancies) and Executioner (dispatches all the expressions into subqueries and sends them to the appropriate sources) will in turn process the mediated query and obtain the *extensional answer* for the user. The difference between extensional and intensional answers is that extensional answers give the fact-sets (the data) which one normally expects of a database retrieval, whereas intensional answers give the “explanation” of what type of answers can be retrieved *without* actually retrieving data from the sources. In a sense, an intensional answer is a characterization of the extensional answer.

## Transformation to an Abductive Framework

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

From observing  $A$  and the axiom  $B \rightarrow A$  ( $B$  leads to  $A$ )

Infer  $B$  as a possible “explanation” of  $A$ .

Abduction is typically used for identifying *explanations* for *observations*. It is a form of non-monotonic reasoning, because explanations which are consistent with one state of a knowledge base (theory) may be inconsistent with new information [Goh, 1996]. For instance, given two axioms ( $B_1 \rightarrow A$ ;  $B_2 \rightarrow A$ ): 1. “It rained”  $\rightarrow$  “the grass field is wet,” and 2. “sprinkler was on”  $\rightarrow$  “the grass field is wet”, the observation that “the grass field is wet” will lead us to conclude that “it rained” or that “the sprinkler was on.” In order to ensure consistency, if we have already known that it did not rain, we have to retract the explanation that “it rained.” We can see that the remaining explanation “sprinkler was on” does not necessarily explain the situation because we did not include all the possible explanations.

Abductive reasoning has been used in diagnostic reasoning (e.g., as in medical diagnosis where observations are symptoms to be explained), natural language understanding, and database view updates [Kakas et al., 1993]. As in the case of **Context Interchange**, the Context Mediator deduces an *abductive solution* in the form of a mediated query, based on the “observation” of the contexts of the data sources. According to Goh [1996], all semantic-comparisons are performed in the context of the user issuing the query. In other words, in comparing two values (e.g. company names, revenues) originating from two different sources, the values are said to be “identical” if they have the same value in the context of the user. For example, the ticker name “HMC” used in Yahoo and the name “Honda Motor CO” in Disclosure both represent the same company: Honda.



Following Eshghi and Kowalski [1989], we define an abductive framework, as seen in Figure 2-7, to be a triple  $\langle T, I, A \rangle$ , where  $T$  is a theory,  $I$  is a set of integrity constraints, and  $A$  is a set of predicate symbols, called *abducible* predicates. Given the COIN framework  $F_C = \langle D, E, C, S, \mu \rangle$ , this can be mapped to a corresponding abductive framework  $F_A$  given by  $\langle T, I, A \rangle$  where

- $T$  is the Datalog<sup>neg</sup> translation of the set of expressions given by  $E \cup D \cup C \cup \mu$ , where the symbol ‘ $\cup$ ’ means *in union with*, or *together with*;
- $I$  consists of the integrity constraints defined in the source set  $S$ ; and
- $A$  consists of the extensional predicates defined in  $S$ , the build-in predicates corresponding to arithmetic and relational (comparison) operators, and predicates corresponding to externally defined functions.

Consider, for instance, the theory  $T$  that summarizes a set of events based on their observations:

$$T = \{ \textit{broken spokes} \rightarrow \textit{floundering wheel}. \\ \textit{flat tire} \rightarrow \textit{floundering wheel}. \\ \textit{punctured tube} \rightarrow \textit{flat tire}. \\ \textit{leaky valve} \rightarrow \textit{flat tire}. \}$$

where  $A = \{ \textit{broken spokes}, \textit{punctured tubes}, \textit{leaky valve} \}$  is the set of predicates from Source  $S$ . Given the abductive query  $Q = \textit{floundering wheel}$  (an example of an observation), we can deduce the following three abductive answers as the *abducible explanations* for the given observation:

$$\mathbf{\Lambda}_1 = \{ \textit{punctured tube} \}$$

$$\mathbf{\Lambda}_2 = \{ \textit{leaky valve} \}$$

$$\mathbf{\Lambda}_3 = \{ \textit{broken spokes} \}$$

Note that “flat tire” is excluded because it is led by other explanations.

Given an abductive framework,  $F_C = \langle T, I, A \rangle$  and a sentence (the observation), the Context Mediator should be able to provide an explanation for the observation derived from:

$$T \cup \mathbf{A}$$

under the integrity constraints that:

$$T \cup \mathbf{A} \text{ must satisfy } I \text{ in } \langle T, I, A \rangle.$$

Thus, the abductive answer  $\mathbf{A}$  can be transformed to the intensional answer as the mediated query<sup>5</sup>.

### 2.3.2 Illustrative Example: Automatic Detection and Reconciliation of Conflicts by the Context Mediator

In this section, we use the integration scenario from section 2.2 to illustrate the computation of the intensional answer to a query. Let us first summarize the context assumptions inscribed in the three web sites and user's preference for the output:

	ScaleFactor	Currency	DateFormat
Disclosure (d)	1000	USD	American "/"
Worldscope (w)	1	Local	American "/"
Olsen (o)	N/A	N/A	European "/"
Users's Preference (u)	1	USD	American "/"

Figure 2-9 Assumptions of Sources and User's Preference

The above figure shows that the user prefers that all the money amounts queried through the system be in US Dollars with a scale factor of one. Of course the user is allowed to have other choices such as thousands of German Marks (GEM) or millions of Japanese Yen (JPY).

First let us start out with a simple query. Suppose that we want to know the net income for the US companies (with revenue greater than US \$1 billion) listed in

<sup>5</sup> This thesis gives a brief explanation on the abductive reasoning behind Context Mediation. For more details on how to derive the mediated query under the COIN framework, please refer to *Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Systems* by Cheng Hian Goh [1996].

Disclosure. Without any knowledge of contextual assumptions in Disclosure, one would enter the following query (Q1):

```
Q1:  SELECT d.companyName, d.netIncome FROM d
      WHERE d.netSales > 1000000000;
```

Without using the Context Interchange system, the query would have only retrieved any company that had sales of more than US \$1 trillion (that's 1,000,000,000,000) from the database because all the recorded sales numbers in Disclosure are in thousands of US Dollars (that means each has been divided by 1000). To correct this and to show the result using a scale factor of one, the Context Mediator would produce the following mediated query (MQ1) as the intensional answer:

```
MQ1:  SELECT d.companyName, d.netIncome * 1000 FROM d
      WHERE d.netSales * 1000 > 1000000000;
```

In order to achieve this, the Context Mediator must have observed from the COIN framework that `netIncome` and `netSales` in the context of Disclosure are using a scale factor of 1000, and have transformed them in the user's context.

As another example, we extend into the use of multiple sources, namely Worldscope, and Olsen. Suppose we want to list the names and net sales of the Japanese companies found in Worldscope, with the condition that these companies have made a net income of at least US \$100 million during the last reported fiscal year. We enter the following query:

```
Q2:  SELECT w.companyName, w.netSales FROM w, o
      WHERE w.currency = 'JPY' and w.netIncome > 100000000;
```

and would obtain the following result illustrated in Figure 2-10:

Source: Worldscope

companyName	fiscalDate	netIncome	netSales	currency
'TOYOTA MTR CORP'	03/31/97	42,879,555,560	1,360,426,111,000	'JPY'
'FORD MTR CO'	12/31/96	4,450,000,000	147,000,000,000	'USD'
'SUZUKI MTR CO LTD'	03/31/97	9,845,342,500	562,325,433,200	'JPY'
'HONDA MTR CO LTD'	03/31/97	29,393,227,200	588,479,835,800	'JPY'

Queried Results (without Mediation)

'TOYOTA MTR CO LTD'	1,360,426,111,000
'SUZUKI MTR CO LTD'	562,325,433,200
'HONDA MTR CO LTD'	588,479,835,800

Figure 2-10 Queried Results from Worldscope without Mediation

We found that this result has some error but it may not look obvious to the user who submitted the query. First, Suzuki's net income (in US\$ is \$89million) is actually less than US \$100 million when calculated using the rate: US \$1 = JPY ¥111. Second, although the values for net sales are correct in Japanese Yen, they are not presented in the user's preference. This might confuse the user in some sense. In order to resolve all the possible conflicts, the Context Mediator generates the following query.

```

MQ2:  SELECT w.companyName, w.netSales * o.rate FROM w, o, u
        WHERE w.currency = 'JPY' AND o.date = '31/03/97'
        AND w.currency = o.fromCur AND u.currency = o.toCur
        AND w.scaleFactor = u.scaleFactor AND w.netIncome * o.rate > 100000000;
UNION
        SELECT w.companyName, w.netSales * o.rate * w.scaleFactor / u.scaleFactor FROM w, o, u
        WHERE w.currency = 'JPY' AND o.date = '31/03/97'
        AND w.currency = o.fromCur AND u.currency = o.toCur
        AND w.scaleFactor < > u.scaleFactor
        AND w.netIncome * o.rate * w.scaleFactor / u.scaleFactor > 100000000;
    
```

Notice there are two sets of expressions. They are basically doing the same thing, but the second part is trying to take care of the difference in scale factors. But in this case, as seen in Figure 2-10, both Worldscope's (w) and user's contexts (u) are using the same scale factor of one. So the second part is redundant in this case and should be removed by the Optimizer. The conditions in the "where" clause make sure that:

- All companies listed are using a currency of 'JPY', meaning that their origin is Japan;
- Their net income, adjusted by the exchange rate, must be greater than US \$100 million;

- The date for the exchange rate must correspond to the end of fiscal year (translated to format in Olsen) for the companies.

The summary of results and the exchange rates and the system uses are listed in Figure 2-11:

fromCur	toCur	date	rate
'Japanese Yen'	'US Dollar'	31/12/96	.0086
'Japanese Yen'	'US Dollar'	31/12/97	.0092
'Japanese Yen'	'US Dollar'	31/03/97	.0090

Queried Results (with Mediation; the net sales below are transformed to US dollars)

Company with Sales > US \$100 million	netSales / rate
'TOYOTA MTR CO LTD'	12,243,835,000
'HONDA MTR CO LTD'	5,296,318,522

Figure 2-11 Mediated Queried Results using Exchange Rates from Olsen

## 2.4 Domain Modeling in COIN


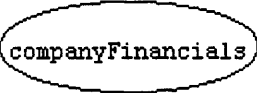
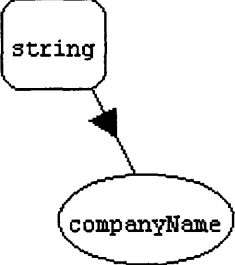
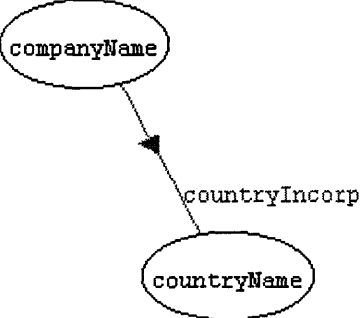
As part of the basis for reconciliation of data by the Context Mediator, the **Domain Model** presents the definitions for the *types* of information units (called semantic-types) that constitute a common vocabulary for capturing the semantics of data in disparate systems. The Domain Model used in **Context Interchange**, or **COIN**, is a “logical” data model that uses mathematical logic as a way for representing data and a language for expressing operations on an underlying data structure. It is also an “object-oriented” data model that supports object-identity, type hierarchy, inheritance, and overriding methods.

The main purposes of deriving the Domain Model for the Context Mediator are to:

1. Describe the *semantic objects* and *relations* presented in the context of the sources, e.g. databases and web sites.
2. Map the data: through the definition of *Elevation Axioms*.
3. Define methods for converting the data from one context to another with the definition of *Context Axioms*.

## 2.4.1 Structural Elements of the Domain Model

Here are the notions that represent the components in the Domain Model of the current system<sup>6</sup>:

	<ul style="list-style-type: none"> <li>• <b>System Node</b> – A rounded rectangle denoting a primitive data type (or system type) from which the semantic type inherits. The most common system types are <i>String</i> and <i>Number</i> (could be an integer or non-integer).</li> </ul>
	<ul style="list-style-type: none"> <li>• <b>Semantic Node</b> – An ellipse denoting a semantic type. It is a derived data type that must inherit from a system type or another semantic type. It represents a piece of meaningful data, such as “<i>companyFinancials</i>”, which can be mapped as the asset value, the profit, or yearly sales of a company.</li> </ul>
	<ul style="list-style-type: none"> <li>• <b>Inheritance Link</b> – A dark line with a big solid arrow showing the relationship between a system type and its subtype, i.e. the semantic type. The direction of the arrow shows that the semantic type inherits from the system type. There is no name attached to the link because it simply describes the inheritance relationship between the system type and the semantic type.</li> </ul>
	<ul style="list-style-type: none"> <li>• <b>Attribute Link</b> – A red line, with a small solid arrow and the attribute name, joins two nodes and denotes that one semantic type obtains the attribute value from another semantic type. For example, a semantic type defined as “<i>companyName</i>” of type <i>String</i> represents the name of a company. It has an attribute named “<i>countryIncorp</i>”, and the value comes from another semantic type called, “<i>countryName</i>” (also of type <i>String</i>) which gives the name of the country that the company belongs to.</li> </ul>
	<ul style="list-style-type: none"> <li>• <b>Modifier Link</b> – A blue line with an outlined arrow and the modifier name. This type of link shows that the value returned from a data source has to be modified</li> </ul>

<sup>6</sup> There are different ways to represent an object-oriented model. One convention, used by Object Management Group, an organization approved by vendors like Microsoft, IBM, and Oracle, discusses about the agreeable standards on the representation. Please refer to <http://www.omg.org/> for more details.

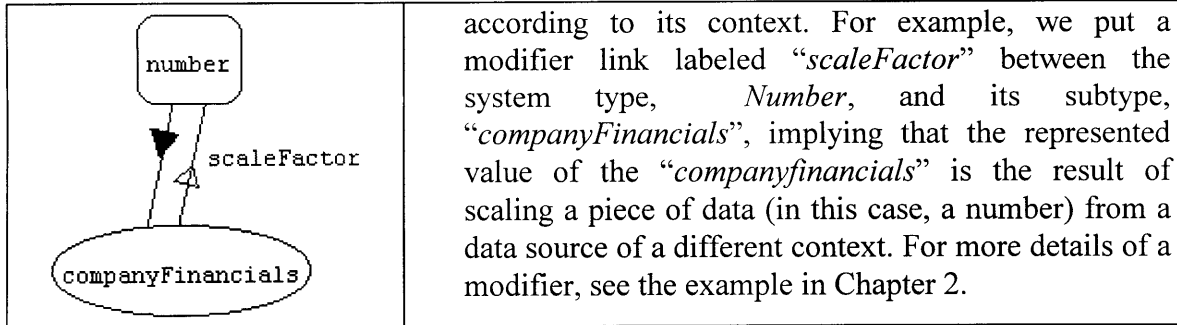


Figure 2-12 Graphical Representation of Different Components in the Domain Model

### System and Semantic Types

As seen in Figure 2-11, the **System type** represents a primitive data type in the Context Interchange system. The most common examples are: *string* and *number*, which can be an integer, float, double, real, etc. The **Semantic type** in Context Interchange is a derived type as in any object-oriented model. It inherits from the System type as a subtype. For example, the name of the company can be defined as a Semantic type which is a subtype of *string* because the name can be best represented as a string.

The definitions of different Semantic types help the system understand the “meaning” of data when they are passed between subsystems or functions within the same system. The instance of the Semantic type is called a *semantic object*, which represents a variable from a data source. For example, an instance of the Semantic type *companyName* can be a variable: *cname* from the table in the Worldscope database and it holds such values as “Oracle Corporation”, “Delta Airlines, Inc.”, etc. Notice that these values may differ from one context to another.

### Inheritance, Attribute, and Modifier

An **Inheritance** link describes the structural and behavioral inheritance relationships between a supertype and its subtype. *Structural inheritance* allows a Semantic type to inherit all the declarations of methods defined for its supertype. Behavioral inheritance allows the definitions or implementations of these methods to be inherited as well. The inheritance in this model can be *non-monotonic*, which means that both declaration and

definition of a method can be overridden in a subtype. This allows the inherited definitions to be changed to reflect any specific data conversion.

An **Attribute** is a structural component assigned to a Semantic object which is an instance of a Semantic type. The purpose of the attribute link in the Domain Model is to associate one semantic object to another. For example, `companyName` has an attribute `country` whose value can be retrieved from the `countryName` object. The data relationships that the attribute links develop help define the elevation axioms for data mapping and deriving extensional relation in a database. The example in next section will explain how to associate one data type with another through the attributes assignments.

A **Modifier** captures any variation in the representation of semantic objects. Every modifier link assigns a conversion function to a semantic object. The main purpose is to derive the value of each semantic object according to the context of the source. It is the basis for context axioms that describes the semantics of individual data elements within sources.



## Chapter 3

# Concepts of Mapping Data to the Domain Model with the PENNY Language

Section 2.4.1 describes the components and the hierarchical properties of the Domain Model. In this chapter, we are going to explore the mapping of data and definition of contexts for the Domain Model. First, we show the construction of the *Domain Model*, which is a feasible model that defines the semantics relation in the integration scenario discussed in chapter 2. It will explain how each piece of financial data from the data sources can be represented by a semantic type in the Domain Model. Then we show how these values can be mapped to the Domain Model via *elevation axioms*. Last, we explain how the user can define the *conversion functions* that can transform the values of the data among different contexts.

To illustrate the definitions of the Domain Model and the elevation and context axioms that correspond to the data sources, we use PENNY for representing the information. As discussed in chapter 1, PENNY is a simplified and easily readable language derived from COINL [Pena, 1997]. One advantage of using PENNY to describe the definitions is that the Domain Model Editor is compatible with PENNY. Thus, it allows a user to understand how to use the Domain Model Editor for defining contexts more easily. For a complete user guide on the user interface and functions of the Domain Model Editor, please refer to Appendix A: “User’s Guide and Programmer’s Reference.”

### 3.1 Defining Semantics Relations in the Domain Model

With the integration scenario exercise at hand, we design a Domain Model that describes the semantics relationships for the underlying data sources. Figure 3-1 shows the TASC Domain Model that helps the Context Mediator perceive the information needed to do the integration:

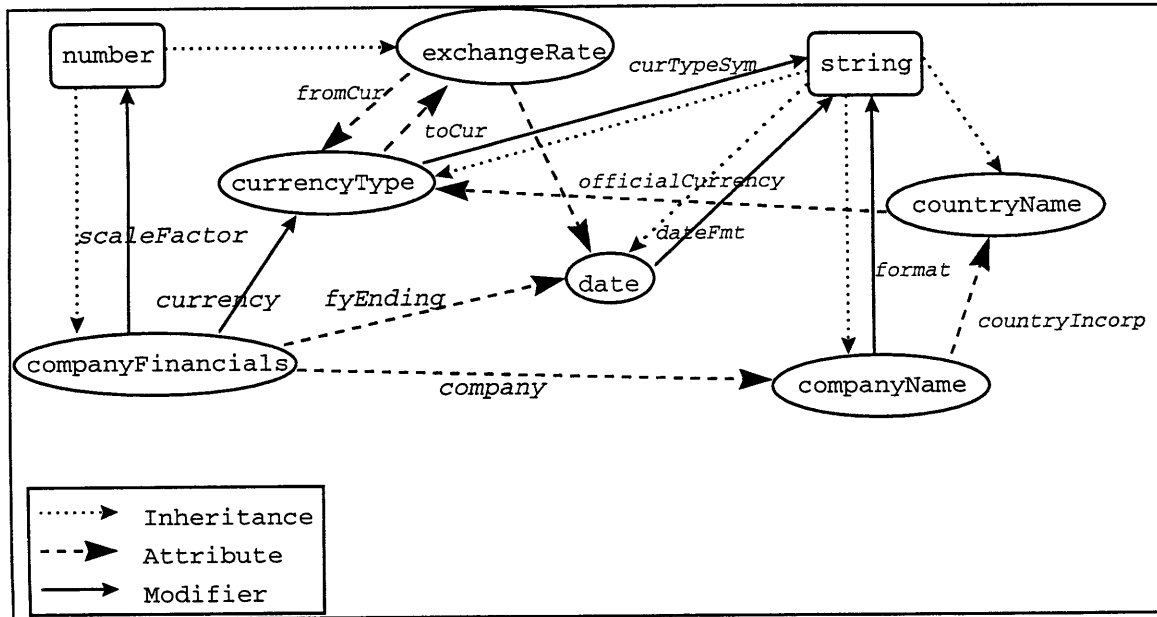


Figure 3-1 The Domain Model for the Integration Scenario

From the diagram, we can clearly see how each data type is related to each other. As described in chapter 2, each oval represents a Semantic type, and each rectangles represents a System type. Every piece of data retrieved from the sources will be mapped to one of these types. For example, all the values regarding the company financials are declared as “companyFinancials” and they become the instances of that Semantic type.

To explain the relationship between each data type, let us start by looking at the Inheritance link. All the Semantic types must inherit from a super type, which can be a System type or Semantic type. For example, “companyName” and “countryName” inherit from type “string” because they represent the name of the company and of its country respectively. The second type, the Attribute links, assigns an appropriate attribute value to the Semantic type. For example, as we know from the Disclosure and Worldscope data in

Figure 2-5 that the net sales or the net income, both instances of “`companyFinancials`”, are associated with the company and the fiscal year end date. So we assign the attributes “`company`” and “`fyEnding`” to “`companyFinancials`.” Similarly, the relation for Olsen requires that the exchange rate be associated with the reference date, source currency, and target currency, so we assign appropriate attributes to “`exchangeRate`” to depict the relationships.

Next, we look at the Modifiers that are responsible for defining the conversion functions during the integration. Notice that each Modifier link in the Domain Model are only labeled with a name. All the required conversion functions are defined separately in the set of Context and Elevation Axioms for each source (in this case, there will be three sets because we have of three sources, each having its own context). Take the Semantic type “`date`” for example, when Context Mediator needs to evaluate the U.S. Dollar amount of Honda’s net income represented in Japanese Yen from Worldscope (as seen in Figure 2-5), the Context Interchange system needs to look up the exchange rate at some particular date. Since the date format in Olsen is in “`dd/mm/yy`” and the date format in Disclosure is in “`mm/dd/yy`”, a special conversion function denoted by the Modifier named “`dateFmt`” has to be applied before the Context Mediator convert the net income from Japanese Yen to U.S. Dollar. Another example is the use of the Modifier link to convert the company financials from one context to another regarding the scale factor. From the diagram, we assign the Modifier “`scaleFactor`” to “`companyFinancials`.” For Disclosure, the scale factor is declared as 1000, meaning that the financial values are presented in thousands of dollar units; whereas for Worldscope, the scale factor is declared as 1 because each financial value in the source shows the actual money amount. When the Context Mediator comes to evaluate the net income and net sales from both sources, it will look up the scale factor declared in the Elevation Axioms. The following section will describe how the user should define the conversion functions and return values in the Context and Elevation Axioms.

An equivalent program written in PENNY that describes the same information is shown below:

```

semanticType companyFinancials::number {
    attribute companyName company;
    attribute date fyEnding;
    modifier number scaleFactor(ctx);
    modifier currencyType currency(ctx);
};
semanticType companyName::String {
    modifier string format(ctx);
    attribute string countryIncorp;
};
semanticType exchangeRate::number {
    attribute currencyType fromCur;
    attribute currencyType toCur;
    attribute date txnDate;
};
semanticType date::string {
    modifier string dateFmt(ctx);
};
semanticType currencyType::string {
    modifier string curTypeSym(ctx);
};
semanticType countryName::string {
    attribute currencyType officialCurrency;
};
};

```

Each entry in the program is a semantic type, the body of each contains all the attribute and modifier definitions associated with the type. The inheritance information is denoted by ‘::’ and the super type (e.g. `string`) attached to the semantic type name in the declaration. Notice that each modifier is a function of the *context* ‘`ctx`’ of the source, for example, `Disclosure`. Such definition allows the Context Mediator to evaluate the values corresponded to a particular context when it is deriving the mediated queries.

## 3.2 Mapping Sources to the Domain Model via Elevation Axioms

The mapping of data and data-relationships from the sources to the domain model is accomplished via the elevation axioms. There are three distinct operations which define the elevation axioms [Goh, 1996]:

- Define a virtual semantic relation corresponding to each extensional relation
- Assign to each semantic object defined, its value in the context of the source
- Map the semantic objects in the semantic relation to semantic types defined in the domain model and make explicit any implicit links (attribute initialization) represented by the semantic relation

As discussed in section 2.3, “Transformation to Abductive Framework,” we present the abductive reasoning done by the Context Mediator to derive intensional answers from some observations. In COIN, we use sets of elevation and context axioms to derive the theory *T*, which gives the abductive answer, in the abductive framework that can evaluate the answer from a query (i.e., the observation).

Examining the elevation axioms for DiscAF below, it shall be shown how each of the three criterion above is met.

1. Elevate 'DiscAF'(cname, fyEnding, shares, income, sales, assets, incorp)
2. in *c\_disc*
3. as 'DiscAF\_p'(^cname : companyName, ^fiscalDate : date, ^shares : void, ^netIncome : companyFinancials, ^netSales : companyFinancials, ^assets : companyFinancials, ^incorp : countryName)
 

```

      {
      ^cname.countryIncorp = ^incorp;

      ^income.company = ^cname;
      ^income.fyEnding = ^fyEnding;

      ^sales.company = ^cname;
      ^sales.fyEnding = ^fyEnding;

      ^assets.company = ^cname;
      ^assets.fyEnding = ^fyEnding;

      ^incorp.officialCurrency = ~ curType;

      ~ curType.value = $ <—
      ~ curType = Incorp.officialCurrency,
      Y = Incorp.value;
      'Currencytypes'(Y,$);
      };
      
```

The first thing that needs to be done is to define a semantic relation for DiscAF. Every semantic object is mapped to a semantic type defined in a domain model and any links present in the semantic relation are explicitly established in the beginning of line 3. The first line gives the external relation being elevated. The second signifies in which context the source values are to be defined. Finally, the third line gives the elevated relation name and the elevated semantic objects (preceded by “^”). The variables (e.g. *cname*, *netIncome*) represent the values of the data. We can call these data entities. They are *instances* of the semantic objects. Since *netIncome*, *netSales*, and *assets* all represent money amounts, we can define all of them as the type *companyFinancials*.

There are several things to note in these elevation axioms:

- Not all semantic objects need to be derived from a semantic relation. For example, the object *curType* is not an elevated semantic object. Rather it is a user defined semantic object and has no existence in the relation ‘DiscAF’. These are called virtual semantic objects because they have no grounding in a semantic relation. The compiler will automatically generate unique semantic object-ids and the system types that go along with them.
- Because such definition is user-defined, the Domain Model Editor cannot automatically generate such code and it has to be manually typed in by the user.

Very similarly, the elevation axioms for WorldAF looks like:

```
Elevate 'WorldAF'(cname, fyEnding, shares, income, sales, assets, incorp)
in c_world
as 'World_p'(^cname : companyName, ^fiscalDate : date, ^shares : number,
             ^netIncome : companyFinancials, ^netSales : companyFinancials,
             ^assets : companyFinancials, ^incorp : countryName)
{
^cname.countryIncorp = ^incorp;

^income.company = ^cname;
^income.fyEnding = ^fyEnding;

^sales.company = ^cname;
^sales.fyEnding = ^fyEnding;

^assets.company = ^cname;
```

```

^assets.fyEnding = ^fyEnding;

^incorp.officialCurrency = ~ curType;

~ curType.value = $ <—
~ curType = Incorp.officialCurrency,
Y = Incorp.value;
'Currencytypes'(Y,$);
};

```

The only differences are the declaration names for the elevation set and the context. The differences in their contexts will be seen in next section. Next, we look at the elevation axioms for Olsen:

```

Elevate 'Olsen'(exchanged, expressed, rate, date)
in c_olsen
as 'Olsen_p'(^exchanged : currencyType, ^expressed : currencyType,
             ^rate : exchangeRate, ^date : date)
{
    ^rate.fromCur = ^expressed;
    ^rate.toCur = ^exchanged;
    ^rate.txnDate = ^date;
};

```

The mapping is simpler for Olsen because it maps only four pieces of data directly and requires no conversions.

### 3.3 Defining Context with Context Axioms

The next task is to define the context axioms in PENNY for the two data sources, namely **WorldAF** and **DiscAF**. Context axioms are a set of definitions for the modifiers of each semantic type given in the domain model. The domain model in text format was stored in a file called `mydomain.pny` and one would like to use the information from the domain model to help derive the context axioms, which will be used by the context mediator for reconciliation of conflicted data. The following code shows the definition of the context axioms of **DiscAF**:

```

1. use('home/proac/Penny/examples/thesis/mydomain.pny');
2. context c_disc;
3. scaleFactor < companyFinancials > = ~(1000);
4. currency < companyFinancials > = ~($) <-
    Comp = self.company,
    Country = Comp.countryIncorp,
    CurrencyType = Country.officialCurrency,
    $ = CurrencyType.value;
5. format < companyName > = ~("ds_name");
6. dateFmt < date > = ~("American Style /");
7. curTypeSym < currencyType > = ~("3char");
8. end c_disc;

```

The above code defines `c_disc` as a context object, which consists of 5 modifiers, namely `scaleFactor`, `currency`, `format`, `dateFmt`, and `curTypeSym`. The objects in “<>” are semantic types and the objects in “~(“ ”) are the return values specific for the data source `DiscAF`. The “~” operator is used to create a semantic object. Line 3 illustrates a modifier as an example. The declaration, `~(1000)`, creates a virtual semantic object whose value is initialized to 1000. This object is then assigned to the `scaleFactor` modifier. The virtual object created here has no name or explicit type. The PENNY [Pena, 1997] compiler takes care of generating unique object-id for the virtual object and generating its correct type using the information in the domain model.

```

1. use('home/proac/Penny/examples/thesis/mydomain.pny');
2. context c_world;
3. scaleFactor < companyFinancials > = ~(1);
4. currency < companyFinancials > = ~($) <-
    Comp = self.company,
    Country = Comp.countryIncorp,
    CurrencyType = Country.officialCurrency,
    $ = CurrencyType.value;
5. format < companyName > = ~("ws_name");
6. dateFmt < date > = ~("American Style /");
6. curTypeSym < currencyType > = ~("3char");
7. end c_world;

```



PENNY allows the definition of modifiers explicitly, as shown in the currency modifier in line 4:

```
currency < companySales > = ~ ($) < .  
    Comp = self.company,  
    Country = Comp.countryIncorp,  
    CurrencyType = Country.officialCurrency,  
  
    $ = CurrencyType.value;
```

Line 4 defines the value of the object returned by the currency modifier obtained from the official currency from the country, whose name is retrieved from the company, which is retrieved from the data source.

## **Chapter 4**

# **Design and Implementation of the Domain Model Editor**

### **4.1 Overview of the Context Interchange Prototype**

The previous chapter provides an overview of the Context Interchange Framework and illustrates how to construct the COIN data model for the Context Mediation process. This chapter acts as a programmer's guide which focuses on the implementation of the Domain Model Editor written in Java. Before we go on, let us look at the overall architecture of the Context Interchange System. Figure 3-1 shows the architecture of the Prototype which is being implemented.

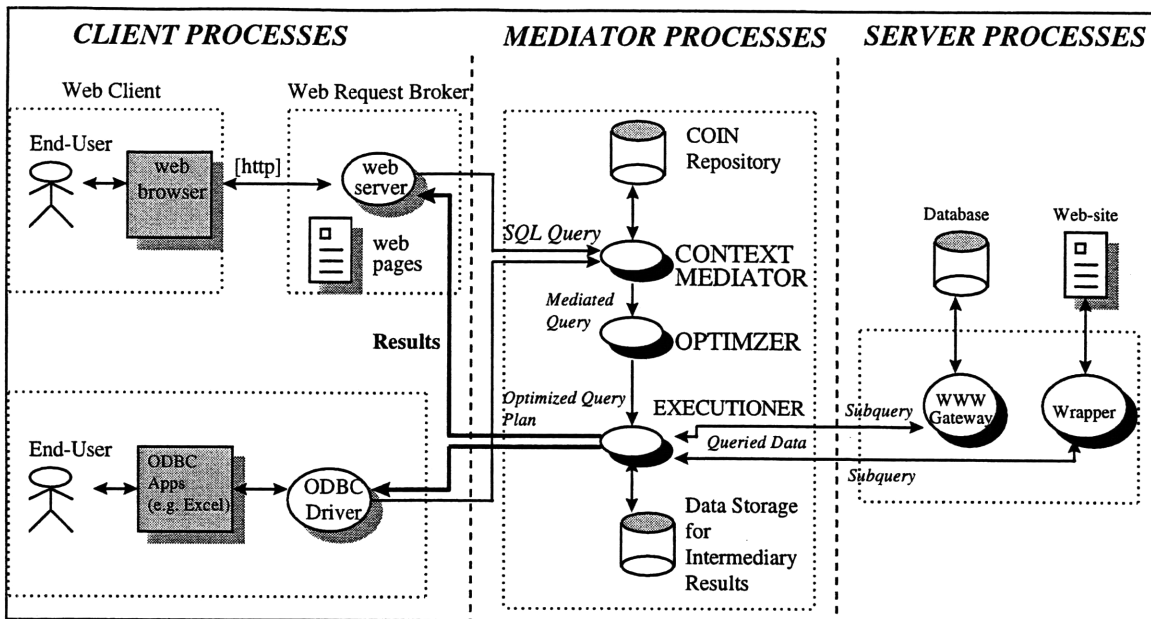


Figure 4-1 Architecture of the Context Interchange (COIN) System

It consists of three distinct groups of processes:

- **Client Processes**

They provide the interaction with receivers and route all database requests to the Context Mediator. Currently the system uses two types of input methods: a CGI script program that produces a web page with forms that allow users to submit queries from a web browser; and an ODBC-compliant application such as Microsoft Excel that issues an SQL query encapsulated as an ODBC request. In the current implementation, this request is intercepted by a custom ODBC driver, which redirects the request to the Context Mediator.

- **Mediator Processes**

The Mediator Processes refer to the system components which collectively provide the mediation services, as described in the first section of Chapter 2. These include:

1. The **Context Mediator** — rewrites a user-query to a mediated query
2. The **Optimizer** — produces an optimal query evaluation plan based on the mediated query

3. The **Executioner** — executes the plan by dispatching subqueries to the Server Processes, collating and operating on the intermediary results, and returning the final answers to the Client)

Currently, these components are being developed and implemented in Java. They are still in the test phase, yet unable to interpret the information from the domain model and context axioms in formulating mediated queries, but read the information from the specification files manually typed in by the developer. In the future, the end-users will be able to design their own Domain Models and let the mediator dynamically generate queries based on the information they provide.

- **Server Processes**

The Server Processes serve as the interfaces that collect data from different outside sources. The components include the **database gateway**, which submits SQL queries to the database servers and returns sets of data from the database tables, and the **wrapper**<sup>7</sup>, which sends an URL consisting of the requested data entry to a web site, reads the web pages generated by the web server, and retrieves the data from the pages. This is accomplished by defining an *export schema* for each of these web-sites, and describing how attribute values can be extracted from the web pages using regular expressions.

## 4.2 Design of the Domain Model Editor

In chapter 2, we showed that the Domain Model, along with the Context and Elevation Axioms, act as a knowledge base of the data sources used by the Context Mediator. Without any help of a graphical tool, such construction process is very difficult to carry out because the user has to spend a lot of time learning how to write the PENNY code to describe all the information. The semantics relations are hard to follow and visualize by just looking at the source code. In addition, the files are stored in a local system which are not easily accessible by many users.

---

<sup>7</sup> See Also: A more detailed description on extracting data from web pages is available in “Data Wrapping on the World Wide Web” by J. F. Qu, 1996.

To solve the above problems, I designed a system, which consists of: 1) the **Domain Model Editor**, a graphical tool which enables the users to define the domain model graphically using a web browser; 2) the **Code Generation Wizard**, which translates the picture and PENNY code back and forth; and 3) the **Remote Method Invocation Tool**, which is a file server that allows users to perform file input/output on the host machine (for saving and loading the Domain Model code).

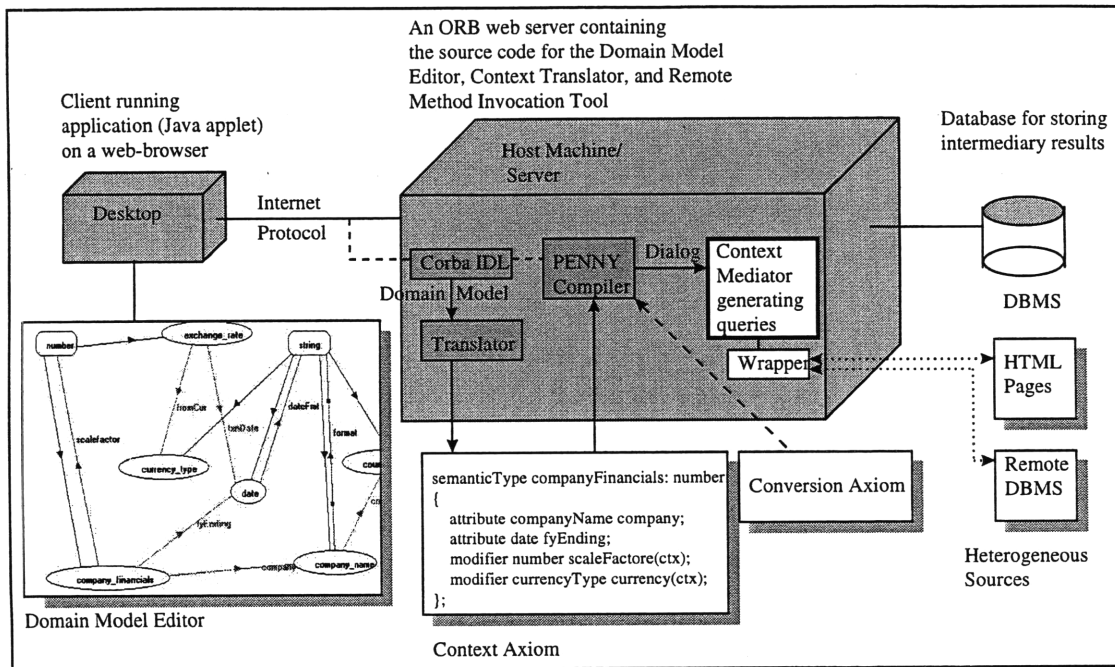


Figure 4-2 The Architecture of the Domain Model Editor System

## 4.2.1 Architecture

Figure 4-2 shows the architecture of the Domain Model Editor System implemented in Java version 1.1. On the client side at the left, the user interface runs as a Java applet that is started in a web browser (such as Netscape 4.0 and Internet Explorer 4.0). The “host machine” shown in the middle of the diagram does the following jobs:

1. Runs a web server that receives any user’s requests through HTTP and sends the applet source code back to the user’s machine, which will in turn run the applet with a Java virtual machine available from the applet-compliant web browser.

2. Runs the Remote Method Invocation Tool (File I/O server) that performs file input/output on the host machine.
3. Contains the PENNY compiler that compiles the Domain Model and context axioms into code that is interpreted by the Context Mediator.

## 4.2.2 Design of the Domain Model Editor — User Interface and Representation of the Domain Model

Figure 4-3 shows the components in the Domain Model Editor:

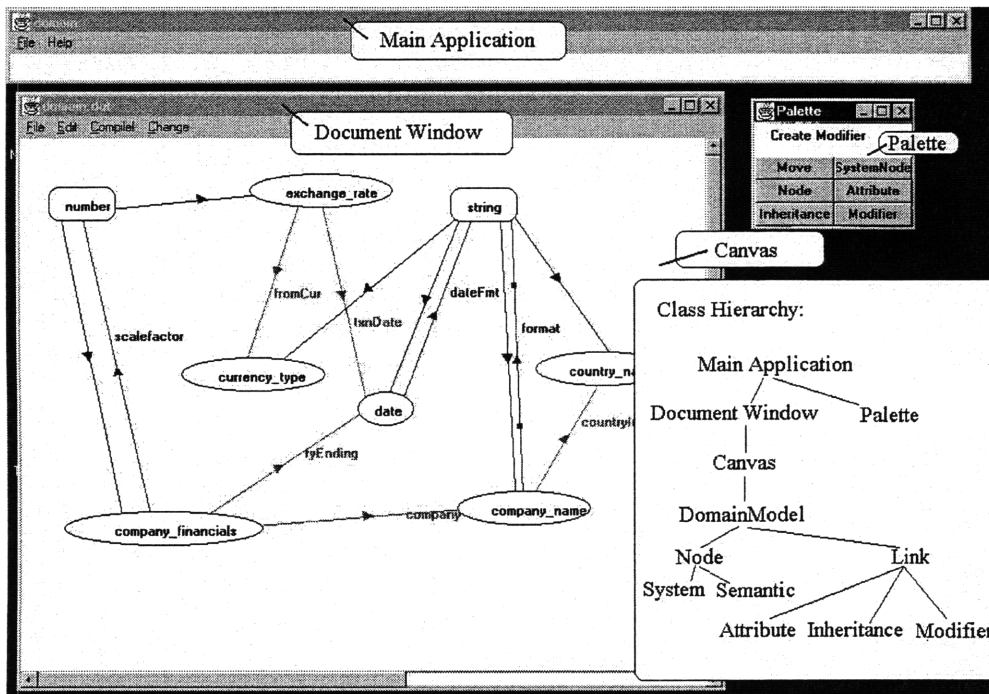


Figure 4-3 Components of the Domain Model Editor

It consists of mainly two parts, namely the user interface components and the Domain Model Components.

## User Interface components

As described in section 1, the user interface components include the Main Window, Palette, Document Window with its drawing Canvas and Code Generation Wizard Box as well as the Main Applet. Each of these components is defined in a separate class:

- **MainApplet** — inherits from `java.applet.Applet` that initiates the program. It receives requests from the web browser which gives calls that start and stop the applet.
- **MainWindow** — the entry point to start the application. The single `MainWindow` object is instantiated within `MainApplet`, or directly by the command:

```
java MainWindow
```

if the user chooses to run the program as a local Java application. This class handles the opening and closing of the `DocumentWindow` and the `PaletteBox`.

- ◆ **PaletteBox** — inherits from `java.awt.Dialog`. As discussed in chapter 1, it contains a set of buttons for drawing a `DomainModel` component corresponding to the label of each button. The `PaletteBox` object is contained within the `MainWindow` class.
- ◆ **DocumentWindow** — each `DocumentWindow` contains a single object of each of the following classes: `DomainModel`, `MainCanvas`, and `CodeDialog` (the Code Generation Wizard Box). It handles local file input/output functions that load and save the files for the `DomainModel` and the codes for the context and elevation axioms. Since the application is a multiple-document interface, the `MainWindow` object keeps a list of `DocumentWindow` objects being created.
  - ◇ **CodeDialog** — subclass of `java.awt.Dialog`, It provides the interface for user's generation of the source code derived from the `DomainModel`. The `CodeDialog` object is instantiated and contained within `DocumentWindow`.
  - ◇ **MainCanvas** — this class takes care of all the drawing of the application. It contains the `paint()` method and the reference to the `java.awt.Graphics` object that repaints the `DomainModel` on itself (the drawing area) whenever the program

requests a redraw. `MainCanvas` contains the `DomainModel` object and triggers it to redraw itself by passing the `Graphics` object to `DomainModel`.

- **MultiLineLabel** — A generic string-drawing class inherited from `java.awt.Canvas` primarily for displaying a string message in Dialog boxes, namely `YesNoDialog` and `MessageDialog` in this application.
- **MessageDialog** — A simple Dialog box displaying a `MultiLineLabel` message used for notifying the user with some important information, such as a warning or error message.
- **YesNoDialog** — A Dialog box that contains a question asking for user's intention before some operation starts to take place. One example is to ask the user whether to save a modified document before closing the `DocumentWindow`. To answer, the user clicks one of the following buttons: "Yes", "No", or "Cancel."

### Domain Model components

- **DomainModel** — the `DomainModel` class is the heart of the application because it contains all the information of the Domain Model. It saves all the Links and Nodes in two instances of class `java.util.Vector`, an implementation that links a list of objects together. One of the `Vector` objects stores a list of `Node` objects: both objects of class `Systemnode` and `Semanticnode`; the other stores a list of `Link` objects: `Inheritance`, `Attribute`, and `Modifiers`. Whenever the program needs to add, remove, draw, cut, save and load a `Node` or `Link` object, it will call the methods in `DomainModel` to perform the operations. It also contains the implementation that translates the picture to PENNY format or does it backward.
- **Node interface** — contrary to a class that can be instantiated, `Node` is declared as an interface, which is an abstract class that declares the methods for the subclass that inherits from it. Its primary function is to allow an object to be cast from a `Vector` list since the retrieved object's class is unknown from the list. It declares several methods that are common to `Systemnode` and `Semanticnode`, such as `getName()`, `getNodeIndex()`,



getType(), etc. The bodies of these methods are implemented separately in class Systemnode and class Semanticnode because each class has its own properties and different ways of responding a method call. For example, when getType() is called, the Systemnode object will return the string “System”, whereas the Semanticnode object will return “Semantic.” Please see more description on the methods in this interface in Appendix A.

- **Systemnode** — it is the class that represents a System Node in the Domain Model. It implements the methods declared in the Node interface Whenever the user requests to create a new System Node, DomainModel will instantiate a new Systemnode object and adds it to the Vector list of Node objects. Whenever the picture needs to be redrawn, it receives a request and the Graphics object handle from DomainModel and draws itself with the methods provided in class Graphics.
- **Semanticnode** — the class that represents a Semantic Node. Similar to Systemnode, this class does the same basic operations such as drawing, returning its type, name, node index, etc. It also contains an integer that stores the index of the node that it inherits from.
- **Link interface** — similar to Node, this interface is the abstract class for all types of links in the Domain Model. For a list of methods declared in this interface, please go to chapter 4.
  - ◆ **Inheritance** — a class that represents an Inheritance link in the Domain Model. It stores two indexes, one for the super type (in most cases the Systemnode) and the other for the subtype (Semanticnode). Its draw() method draws the link based on the locations of the two nodes it is connecting; and an arrow which shows that the direction pointing to is the subtype, and the other is the super type.
  - ◆ **Attribute** — this class represents an Attribute link in the Domain Model. It contains the name of the attribute, and the indexes of the Semanticnodes that it is connecting. It also draws an arrow which is pointing the Semantic type that represents the attribute value.

- ◆ **Modifier** — denotes the Modifier link. It indicates that there is a conversion function or value that the Semantic type is applying to its super type. The name that attaches to the Modifier link represents the name of the conversion function or the conversion value (e.g. `scaleFactor` which represents values like 1, 1000, etc.).
- **Arrow** — this class creates a generic drawing object, an arrow. Every Link object initiates and contains an arrow object, whose drawing properties (size and color) depend on the type of Link. For example, an Inheritance link has an arrow which is drawing as a solid black and large triangle, whereas an Attribute link holds an arrow which is drawing itself as a small, red solid triangle, and so on.

### 4.3 Implementation of the Client Application with Java

Because the Domain Model Editor is primarily a client application, more than 90% of the code is written for this part. The rest of it is for the remote File I/O Server using Java RMI. The Java programming language is developed by SunSoft, a division of Sun Microsystems. Its major goal is to make the program portable such that they can run on many different platforms. The criteria that we use to choose Java over other languages for implementing the Domain Model Editor include:

- **Object-oriented design and General Issues**

The primary goal of this application is to construct an object-oriented Domain Model, which consists of hierarchies of semantic data types inherited from primitive data types such as strings and integers. The instances of these types are the data values from the data sources. Such object-oriented structure is best presented in an object-oriented language such as Java and C++, which is probably the most popular object-oriented programming language to date.

Because the Domain Model Editor is very graphically intensive, it requires a lot of hands-on drawing and user interface libraries in support of the program. If only based on the completeness of such libraries, C++ should be the best choice for

implementing this application because many vendors such as Microsoft, Borland, and Symantec have developed sophisticated software that can produce complex applications with appreciable performance. But one has to possibly sacrifice much effort on changing and porting the code to run on a different platform, such as from Windows 3.1 (16-bit) to Window 95 (32-bit), or totally different systems like Sun Solaris or Mcintosh.

The Java language itself (excluding the API functions) in a sense is a subset of C++ because the structure is the same except that it forbids the use of pointers. It makes Java an easier programming language than C++ because it avoids the confusion over the use of different references of identifiers for the instances of classes, but at the same time does not lose much functionality. Java also has improved features such as garbage collection, which reduces the chance of memory leak when the program is running; and exception handling, which returns any detailed message during runtime.

- **Portability/Scalability**

The major advantages of Java over other languages such as C++, PERL, CGI-scripts are its portability and scalability. Java allows program running on many different platforms. This is very important for building web applications such as the components in the Context Interchange System. The network availability allows a user to run the application from a local machine with having to worry about what platform it is running, as long as the user has installed a Java virtual machine that usually comes with the web browser. The programs are downloadable via the Internet and usually it requires loading the application from a browser with a mouse click, so minimum installation is needed.

Unlike PERL or CGI-scripts programs which consists of procedure calls, Java uses objects that is capable of recording states and transactions of an operation. Such feature allows the reproduction of the same object to serve several clients with the same functionality. This enables the server to respond to the clients more efficiently. If one uses PERL or CGI-scripts to implement the program, the web server may have

problems of bottle-necking because it cannot handle multiple clients requesting to run the same procedure at the same time.

- **Extensibility**

Java uses the **RMI** (Remote Method Invocation) model for implementation of distributed applications. Sunsoft is developing a tool that supports the **IIOP** (Internet Inter-Orb Protocol) and can make Java programs compliant with **CORBA**. CORBA is a multi-vendor supported architecture that enables programs of different languages (such as C, C++, PERL, Java) to interact with each other through a standard interface language called **IDL**, or Interface Definition Language. This makes life easier for users who want to integrate their Java programs with other applications that are written in another language.

Visual Basic and Visual C++, developed by Microsoft, allow export of the source code to other Microsoft Applications using the ActiveX technology. Since our project is dealing a variety of languages such as PROLOG on UNIX platforms, we want to allow more flexibility in exporting the code to DEC system, Sun Solaris, not just Windows. As a result, Java is a better choice over Visual Basic and C++ in terms of extensibility.

- **Security**

Because Java applications are often widely used across the Internet, security is important in limiting any unauthenticated use of private data and applications. By default, the web browsers do not allow any local access (i.e. save, load, print, or execute files) to the client machine by any applet downloaded from the host. To enable functions to operate in the host machine safely, users can apply different authentication schemes such as SSL to write an encrypted Java client applet and install an encryption protected web server in the host. This allows the server to have limited access only available through a user password and a specific key.

## **Chapter 5**

### **Conclusion: Limitations and Future Work**

In conclusion, I listed out the problems that I have encountered when implementing the Domain Model Editor system using Java. As of time of writing this thesis, Java is still in an early phase of development. There are still many ways to make Java more robust.

The last section will discuss some of the issues regarding integration of the application in the current system. While most of the components including the Optimzer and Executioner in the Context Interchange systems are still being developed (as discussed in last chapter), they exist as loosely pieces and need time to be integrated together to form one completely functional application. Ideally, we would like the server part of the system to be easily accessible by an administrator or an end-user from a client machine. As an extension, we can have more than one servers running the system at the same time. All these allow the system to run in scalable and global networking environment useful for enterprises and international business.

#### **5.1 Programming with the Java Development Kit**

As of the day this document is written, the latest release of Java Development Kit (JDK) is version 1.1.6. The very first major release of the Java Development Kit is version 1.0.2 back in 1996, and the next release, 1.2, is still in beta version. The following are the limitations found in JDK 1.1 that affect this application:

- **Drawing** — as part of `java.awt` API library, the drawing package does only the basic job of drawing. Although the package is capable of drawing various types of rectangles, circles, and polygons, it does **not** have *shadows*, *arrows*, *slanted fonts*, or any kind of *dotted lines*. In addition, it does not support the *stacked and transparent drawing panes*, which is an important feature for moving the components around smoothly on the picture. It causes flicking when a component on the canvas is being moved. These limitations greatly affects the Domain Model Editor in terms of the drawing features.
- **Graphical User Interface** — also part of the `java.awt` API, the user interface package loosely contains user interface components such as buttons, simple list boxes, choice boxes, and menus. Moreover, the list box and pop up menu classes are known to have problems and do not work properly in JDK 1.1. In summary, the JDK user interface and drawing packages are incompetent as compared to Visual C++ which provides advanced user interface features such as directory tree and database tables. Nevertheless, from the descriptions in the SunSoft web site, the latest version of JDK, 1.2 seems to provide more useful functions.
- **Application Distribution** — As mentioned in last section, SunSoft is planning to make Java applications compatible with applications written in other languages by support IIOP, a protocol being used by CORBA. As of the day of writing this document, there is no tool that can effectively make Java programs work with programs in common languages like C, C++, PERL, CGI, or perhaps PROLOG which is the language that implements the Context Mediator and the PENNY compiler. Currently, vendors such as Visibroker are developing IDL packages under CORBA that support interactions between programs written in C, C++, PERL and Java. Oracle's Networking Computing Architecture is also supporting the languages mentioned above. However, due to relatively rare usage in the market, PROLOG is not reportedly supported. One plausible but very complicated way is to define IDL interfaces under CORBA to allow the Java program to interact with a C program, which in turn communicates with the PROLOG programs through special libraries.

## 5.2 Domain Model Editor

- **Server Capabilities** — currently the server part provides the File input/output services on the host machine. It is incapable of triggering the PENNY compiler to compile the Domain Model because the current implementation of the Context Mediator requires that the server reboot every time to update the new Domain Model. In the future, one can redesign the system such that it can skip the rebooting part and allows the Domain Model Editor to restart the system.
- **Code Generation Wizard** — at this moment the application only supports parsing the PENNY code. Since the PENNY compiler will eventually translate the PENNY code to PROLOG , the Code Generation Wizard should also support the parsing of PROLOG.
- **Defining Conversion Functions** — the user has to manually type in the conversion functions on the editing box. A library of standard conversion functions such as deriving the annuity should be available in the future.
- **“Illegal” Domain Model Checking Rules** — the Domain Model Editor right now does not do a complete checking of all the possible errors in creating the Domain Model. The errors that the program can detect are:

Violation of rules involving Links — Logical Errors

- ◆ a System type cannot inherit from a Semantic type.
- ◆ user can assign an attribute link if and only if both of the Nodes are Semantic nodes
- ◆ user can only assign a Modifier to a Semantic type. The Modifier link must point to the super type that the Semantic type inherits from. These are the errors done by user that cannot be detected by can possibly be detected by the PENNY compiler
- ◆ looping: for example, when System type “A” inherits from System type “B”, the user can create a new System type “C” that inherits from “B” and allows “A” to inherit from “C”. This is a logical error.

- ◆ duplicate names: currently, any two components, Nodes or a Links, can have the same name. This should be illegal and detected.

### **5.3 Integration with the Current COIN Components**

The major interaction between the Domain Model Editor and the rest of the system right now is the passing of output files (Domain Model and context definitions) for the PENNY compiler to compile. The implementation of File Server does the job by saving the files in the server that runs the COIN system. In the future, we can improve the current application by adding a trigger module that starts the compiler. One problem that exists is that the current system has to be restarted whenever the context information is updated. This limits the controllability by the application that is run remotely because the administrator must sit next to the host machine in order to restart the system. We have to investigate more on this. Another issue, as stated at the beginning of the chapter, that we might focus on is the network accessibility by the end-user of the system.

In conclusion, the goal of the Context Interchange strategy is provide novel approach for reconciliation of conflicts in heterogeneous systems. We discussed the fact that it would be a good feature if a user has full access to the system away from the server. This allows the system to run a global environment. In addition, we can design the system in the future such that more than one servers are running. This will speed up the mediation process such that users can utilize the truly powerful features of the Context Interchange System.



# Bibliography

- Bressan, S., Fynn, K., Pena, T., and et al. (1997). Demonstration of the context interchange mediator prototype. In *Proceedings of the ACM SIGMOD/PODS Conference on Management of Data*, Tucson, AZ, May 1997.
- Bressan, S., Lee, T., Goh, C., and et al. (1997). A procedure for context mediation of queries to disparate sources.
- Dobbie, G. and Topor, R. (1995). On the declarative and procedural semantics of deductive object-oriented systems. *Journal of Intelligent Information Systems*, 4:193-219.
- Eshghi, K and Kowalski, R.A. (1989). Abduction compared with negation by failure. In *Proc. 6<sup>th</sup> International Conference on Logic Programming*, pages 234-255, Lisbon.
- Goh, C.H. (1996). *Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Systems*. PhD thesis, Sloan School of Management, Massachusetts Institute of Technology.
- Kakas, A.C. and et al. (1993). Abductive logic programming. *Journal of Logic and Computation*, 2(6):719-770.
- Kowalski, R.A. (1990). Problems and promises of computational logic. In Lloyd [1990], pages 1-36.
- Lloyd, J.W. (1987). *Foundations of logic programming*. Springer-Verlag, 2<sup>nd</sup>, extended edition.
- Madnick, S. (1996). Are we moving toward an information superhighway or a tower of babel? The challenge of large-scale semantic heterogeneity. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 2-8, April 1996.
- Siegel, M. and Madnick, S. (1991). A metadata approach to solving semantic conflicts. In *Proceedings of the 17<sup>th</sup> International Conference on Very Large Data Bases*, pages 133-145.
- Pena, F. (1997). *PENNY: A Programming Language and Compiler for the Context Interchange Project*. Master thesis, Sloan School of Management, Massachusetts Institute of Technology.

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

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

# APPENDIX A

## User's Guide and Programmer's Reference

### Domain Model Editor Version 1.1

Updated: 05/18/98

For Context Interchange System

© 1998 Context Interchange Group  
Sloan School of Management, M.I.T.

# Preface

*User's Guide and Programmer's Reference: Domain Model Editor Version 1.1* is both a how-to and a reference guide on using the Domain Model Editor to 1) create the domain model, and 2) generate the context axioms, elevation axioms, and conversion functions that correspond to the contexts (how the data are represented and related) of the sources. This information allow the **Context Mediator** to identify and resolve the conflicts that may come up when evaluating the data from different sources, namely relational databases and World Wide Web sites. The major goal of the Domain Model Editor is to help define the meanings of data and the relationships between them, given that the context of the source is known.

## Who should read this?

The Domain Model Editor Application along with this document provide a user friendly and easy-to-visualize approaches for users who plan to develop a knowledge base of the data sources for the **Context Interchange System** that resolves data conflicts arriving from data integration. After following the example in this manual and the step by step guide to create the domain model and context axioms, the user will have a basic understanding on how to generate information for the *Context Mediator* that converts data among different contexts. For descriptions on the concepts behind the **Context Mediation Strategy** developed by the **Context Interchange Group** at Sloan School of Managment, M.I.T., please refer to the thesis: "*Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Systems*" written by Cheng Hian Goh, 1996. Also see the next section for a list of more references regarding the studies of Context Mediation.

The Domain Model Editor is targeted for users who wish to apply the Context Mediation concepts to resolve the semantic conflicts among different data sources through the Context Mediation process. The application is perhaps the most useful tool for people who have trouble understanding and writing code in PENNY language that describes the domain model, context axioms, and elevation axioms for use in the context mediation process. With the Domain Model Editor, the user can design the domain model graphically instead of taking time to write the PENNY code. The program can also convert the existing PENNY code to a picture that is 100% equivalent. With the picture of the model, the user can easily visualize how all the data types are represented and related to each other. The user can also create the context axioms and the elevation axioms with the help of **Code Generation Wizard** that comes with the application.

This is a MUST-SEE document for any software developer who is planning to extend this application because this is the complete guide that

describes the design, implementation, limitations, as well as usage of the Domain Model Editor as a Java distributed application.

## More References

For more details on how to theoretically construct the Context Interchange Data Model, please refer to my thesis: “*Derivation of Context Axioms from the Domain Model Editor in a Networking Architecture*” (1998). For information on the PENNY language and the PENNY compiler, please refer to: “*PENNY: A Programming Language and Compiler for the Context Interchange Project*” written by Fortunato Pena, 1997.

## What’s in this?

This document consists of 2 sections.

**Section 1, “Overview of the Domain Model Editor,”** gives a tour on how to use the basic features and user interface of the Domain Model Editor. It describes how to create a pictorial representation of a simple domain model and how to use the *Code Generation Wizard* to generate PENNY code from the picture or to generate the picture from the source code.

**Section 2, “Functional Specifications and Class Descriptions,”** consists of the description of all the classes and functions in the Domain Model Editor 1.1 source code. It is a useful reference for developers who are planning to extend this application and/or use as a prototype to develop another Java client/server application.

## Requirements

For user running the client application:

Any user who would like to run this application as a Java applet must have installed one of the following:

- Internet Explorer 4.0 or above (<http://www.microsoft.com/>)
- Netscape Communicator 4.0 or above (<http://www.netscape.com/>)
- HotJava Browser for Java 1.1.x (<http://java.sun.com/products/hotjava/>)
- Appletviewer included in the Java Development Kit Version 1.1.x or above (<http://java.sun.com/>)
- Or: Any web browser that supports running Java 1.1 applets

Memory: 32MB of RAM or above recommended

Mouse or other types of pointing device required

**For administrator setting up the application on the server:**

The administrator must have installed an HTTP web server that allows the end-user to download the Java bytecodes from the server to the client machine. The server must also contain Java Development Kit 1.1.x or above in order to start the server daemon.

Memory: 32MB of RAM or above recommended

Hard Disk Space:

- Web Server: Software dependent
- Java Development Kit 1.1.x - approximately 20MB downloadable from <http://java.sun.com/>
- Domain Model Editor - 2MB, including the whole package and full documentation.

**For developer:**

The current version of the Domain Model Editor is written and compiled using Java Development Kit version 1.1.5 for Windows 95/NT from SunSoft (<http://java.sun.com/>). It is compatible with JDK version 1.1.x or 1.2.

## Request for Comments

If you find any errata in this guide, or would like to report a bug or suggest better features for the next version of the Domain Model Editor, please feel free to let me know. Your response will be valuable to the future development of the Context Interchange Project. The best way to contact me is by email at [proac@mit.edu](mailto:proac@mit.edu) or visit our web site at <http://context.mit.edu/>.

## A1. Domain Model Editor

---

### IN THIS SECTION

- Getting Started
- Using the Graphical User Interface
- Graphical Representation of the Domain Model
- Using the Code Generation Wizard
- Saving and Loading Different Formats of Documents

## Getting Started

Because the Domain Model Editor is implemented in Java 1.1, you must obtain a web browser or the Java Development Kit that supports running Java applets version 1.1.x or above. Many web browsers capable of running Java 1.1.x applets are downloadable from the web (some license restrictions or charge may apply):

1. Internet Explorer 4.0 for Windows 95/NT/3.1  
<http://www.microsoft.com/>
2. Netscape Communicator 4.0 for Solaris 2.4, 2.5, Windows 95/NT  
<http://www.netscape.com/>
3. SunSoft HotJava 1.1.2 for Solaris 2.4, 2.5, Windows 95/NT  
<http://java.sun.com/products/hotjava>

If you are planning to develop your own Java 1.1.x application, you can download the Java Development Kit from <http://java.sun.com/products/jdk/1.1>.

Upon completion of the installation of the required software, you will now be able to run the Domain Model Editor. Launch your web browser and type the following URL:

<http://soloaudio.mit.edu/java/domain11/start.html>

The final release version will be available at:

<http://context.mit.edu/~coin/>

You can have the option of downloading the entire Domain Model Editor program and examples onto your own machine so that you can free use the local save and load functions to access the files in your own machine (*Because of internet security, a Java applet downloaded from a web site has limited usage of save/load/print functions in the local machine from which the user is opening the web site. This ensures that the applet from the host cannot harm any client machine that is downloading it. Please see Chapter 3 for more details on the limitation of the Domain Model Editor*).

Although not necessary, you can download the entire Domain Model Editor program such that you can use the save and load features in your local machine. To download, enter the following URL in your web browser:

<http://soloaudio.mit.edu/java/domain11/domain11.zip>

Choose the “Save” or “Save As” option and save the file in a new directory in your hard disk. For example: `c:\java\domain11` in Windows or `/home/proac/java/domain11/` in Unix. Unzip the file and you should see the following directory structure and files:

`c:\java\domain11\start.html`



```
\Readme.txt + {a number of .java and .class files}
\docs\Allnames.html
  \{a bunch of other .html help files}
\out\
```

The start.html file in the new directory is the one that the web browser opens in order to start the applet. The README.TXT contains information on how to compile and run the source code. If you have already installed the Java Development Kit, you could run the following commands:

```
> cd javadomain11           - change to the program's directory
> javac *.java             - recompile the files using your own JDK
```

To run, open start.html in your browser, or type at the prompt:

```
> appletviewer start.html  - run as an applet
or
> java MainWindow         - run as an application (equivalent)
```

## Using the Graphical User Interface

The Domain Model Editor will pop up as three windows: **The Main Editor Window**, an empty **Document Window** entitled, "Untitled\_1", and the **Palette Box**.

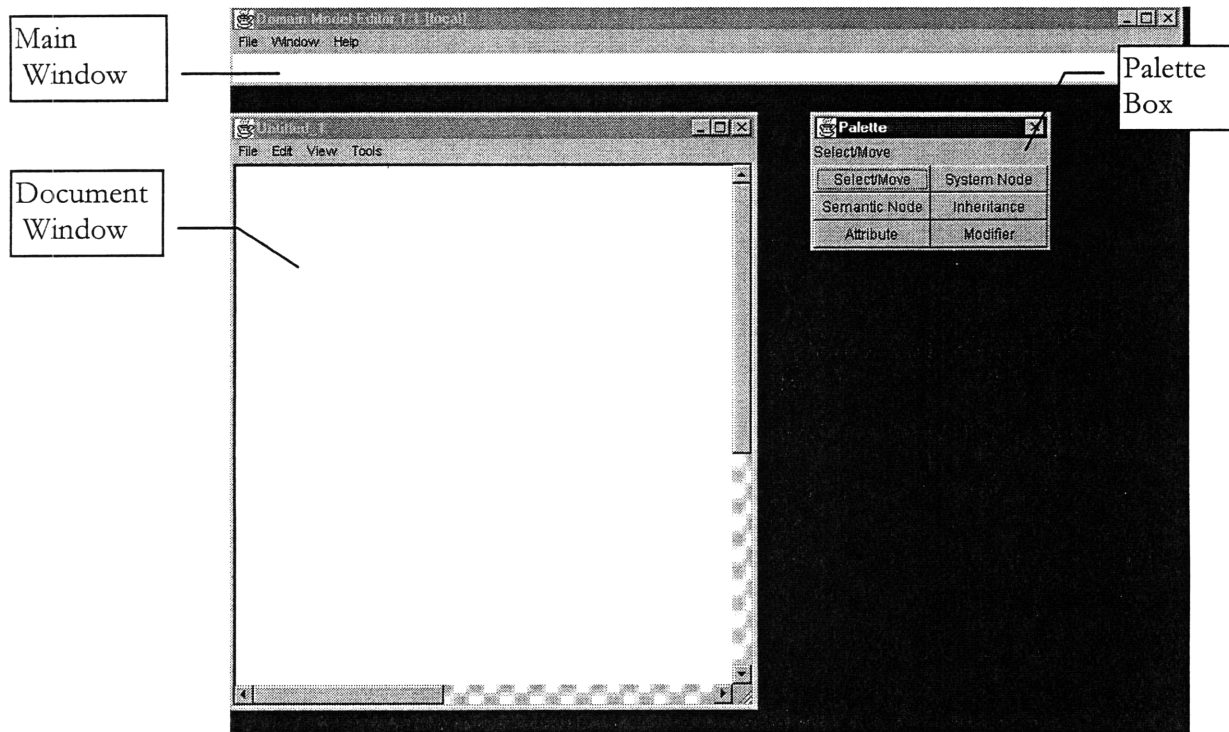


Figure A-1 The Domain Model Editor

As shown in Figure A-1, the **Main Window** on top of the screen consists of a Menu Bar. Its primary functions are to launch a new Document Window, open an existing document, Show/Hide the Palette Box and exit the program. The “**Help**” Menu allows you to view the help content.

The **Document Window** consists of the Menu Bar and the *Canvas* in which the picture of the Domain Model is drawn. The menu contains the following functions:

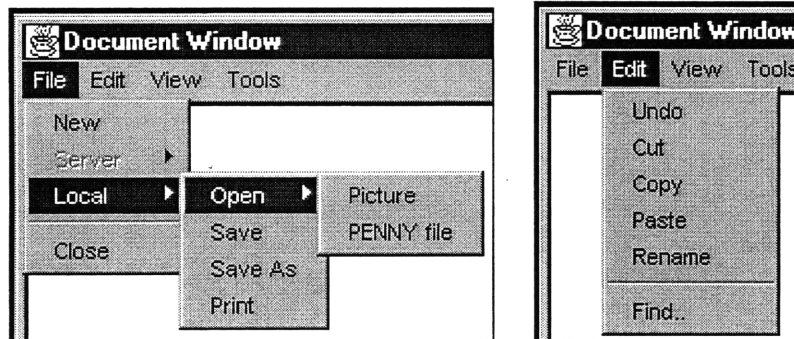


Figure A-2 The **File** and **Edit** Menus

1. **File**

- New — Creates a new Document Window
- Open — Opens a new Document (picture or code) in a new window
- Save, Save As — Saves the picture information in ascii text format
- Print — Prints the picture
- Close — Closes the current Document Window. Asks user if the file needs to be saved before closing.

2. **Edit**

- Undo — Cancels last creation of components, cut, or move.
- Copy — Copies the select component: a node or a link
- Paste — Pastes the component (node only) that was cut or copied
- Rename — Renames the selected component
- Find — Locates and highlights a component

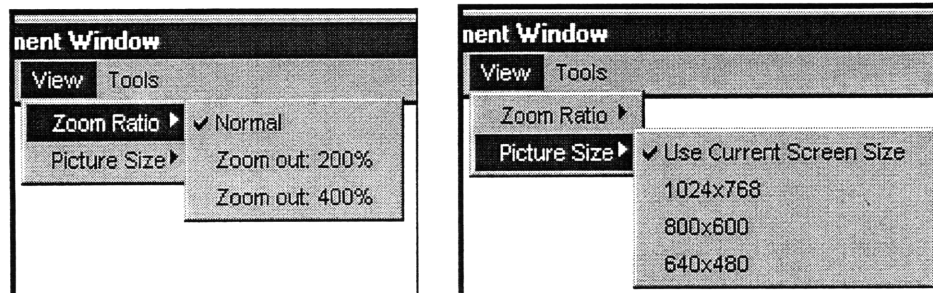


Figure A-3 The View Menu

### 3. View

Zoom Ratio — Selects the zoom ratio of the picture

Picture Size — Selects the size of the canvas


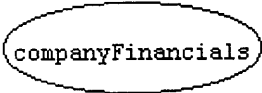
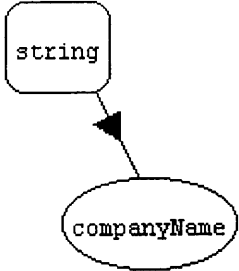
### 4. Tools

Code Generation Wizard — Opens or Closes the Code Generation Wizard Box

The **Palette Box** allows you to select which component to draw. The first button labeled “Select/Move” enables you to highlight a component, move it, and/or change its name. The other five buttons allow you to create the nodes and links on the canvas. The following section describes what each component means and how to create them.

## Graphical Representation of the Domain Model

Here are the notions that represent the components in the Domain Model:

	<ul style="list-style-type: none"><li>• <b>System Node</b> – A rounded rectangle denoting a primitive data type (or system type) from which the semantic type inherits. The most common system types are <i>String</i> and <i>Number</i> (could be an integer or non-integer).</li></ul>
	<ul style="list-style-type: none"><li>• <b>Semantic Node</b> – An ellipse denoting a semantic type. It is a derived data type that must inherit from a system type or another semantic type. It represents a piece of meaningful data, such as “<i>companyFinancials</i>”, which can be mapped as the asset value, the profit, or yearly sales of a company.</li></ul>
	<ul style="list-style-type: none"><li>• <b>Inheritance Link</b> – A dark line with a big solid arrow showing the relationship between a system type and its subtype, i.e. the semantic type. The direction of the arrow shows that the semantic type inherits from the system type. There is no name attached to the link because it simply describes the inheritance relationship between the system type and the semantic type.</li></ul>

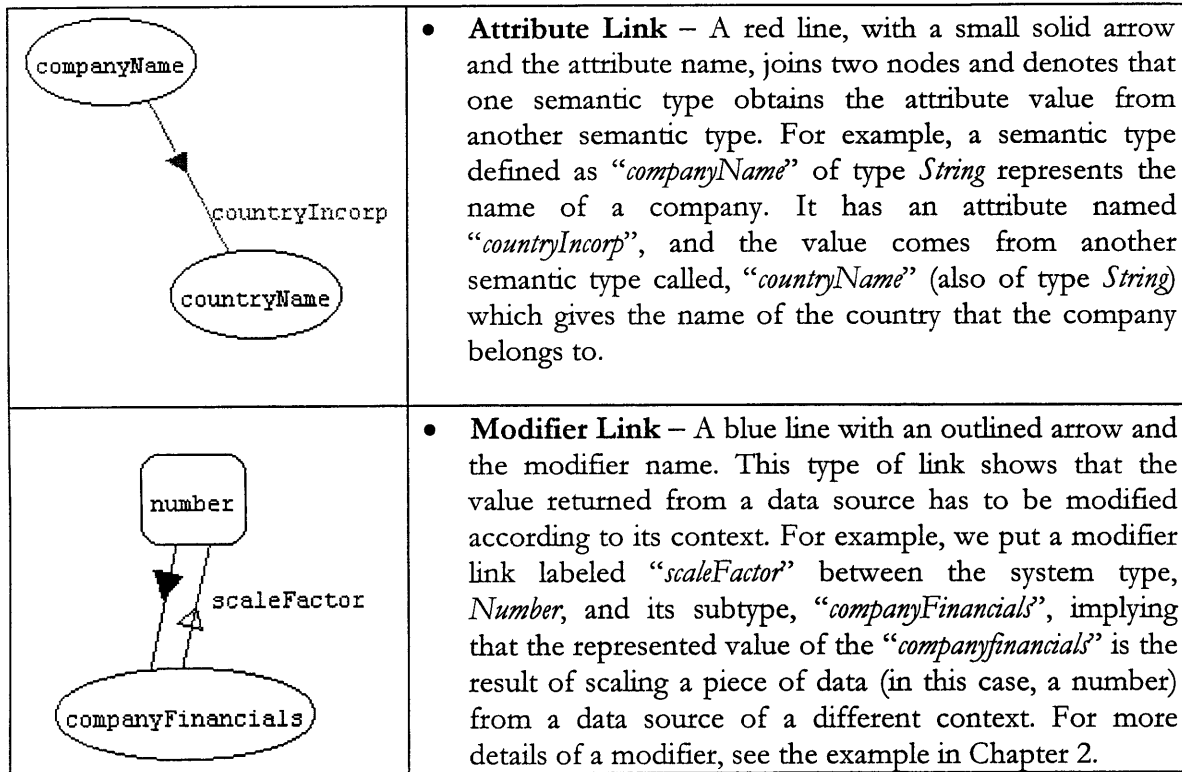


Figure A-4 GraphicalRepresentationof Different Components in the Domain Model

The above description might be way too brief to fully explain the concepts of mapping the data into a domain model. Nevertheless, you can now start drawing a simple Domain Model as a way to get familiar with the drawing tool. The drawing tool is fairly simple to use.

Before drawing any link you must start by drawing a system node or a semantic node. Try clicking the “System Node” button on the Palette box, then click on the white area (Canvas) of the Document Window. A system node with the default name “(0)” will show up. The number “0” means that it is the very first node that you create. Try creating a semantic node by clicking the “Semantic Node” on the palette box and then clicking on the canvas of the same document window. A semantic node labeled “(1)” will show up on the screen. The next created node will labeled “(2)” and so on, until you change the name of any of the nodes.

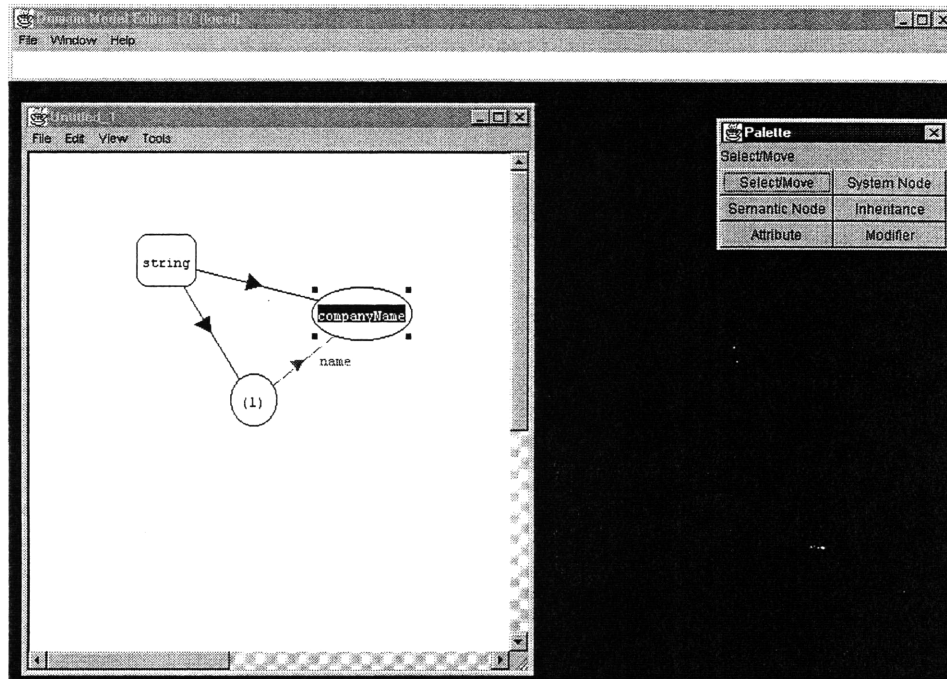


Figure A-5 Creating new system and semantic nodes and changing the names

To change the name of a node, click the “Select/Move” button on the Palette box, and then double-click the node. The name will be highlighted and you can start typing in the new name. Press “Enter” on the keyboard to denote finishing typing the name or press “Esc” to resume the name before the change. Let’s call the system node “*String*” and the semantic node “*companyName*”. Notice that the program will not accept a space or any other characters except for the alphabets and numbers. This is because all the names in the picture will be used by the compiler for declaration of data types. An empty space or a symbol other than the underscore: “\_” would possibly cause a compiling error.

Now you can try creating an inheritance link, pointing from the system node labeled “*String*” to the semantic node labeled “*companyName*”. As mentioned, an inheritance link must point from a type to its subtype. Because *String* is a primitive system type, we can only join an inheritance link from the system node to the semantic node and not the other way around. You can possibly define a semantic type that inherits from another semantic type, but not a system type that inherits from a semantic type because it violates the object-oriented modeling rule. Doing so would result in getting a warning signal from the program.

## Using the Code Generation Wizard Box

The main purposes of the Code Generation Wizard Box are:

1. Converting from picture to code:

- Converts the graphical representation of the Domain Model to the equivalent domain model in PENNY format.
  - Based on the modifiers information provided in the picture, generates the context axioms code in PENNY format.
2. Converting from code to picture:
- Converts the PENNY domain model to the graphical representation.

(Current version (1.1) only supports the PENNY format. The next update will include PROLOG, a common program-ming language that the PENNY domain model will be convert-ed to eventually)

The functions provided in the Code Generation Wizard Box allows users to convert back and forth the picture and source code. This makes it easier for users who want to view the semantic relationships of the existing code written in PENNY, and also for those who would like to create a new model without writing any code.

To open the Code Generation Wizard box, click on the menu: “Tools” of the Document Window’s menu bar, then click on the menu item: “Code Generation Wizard:

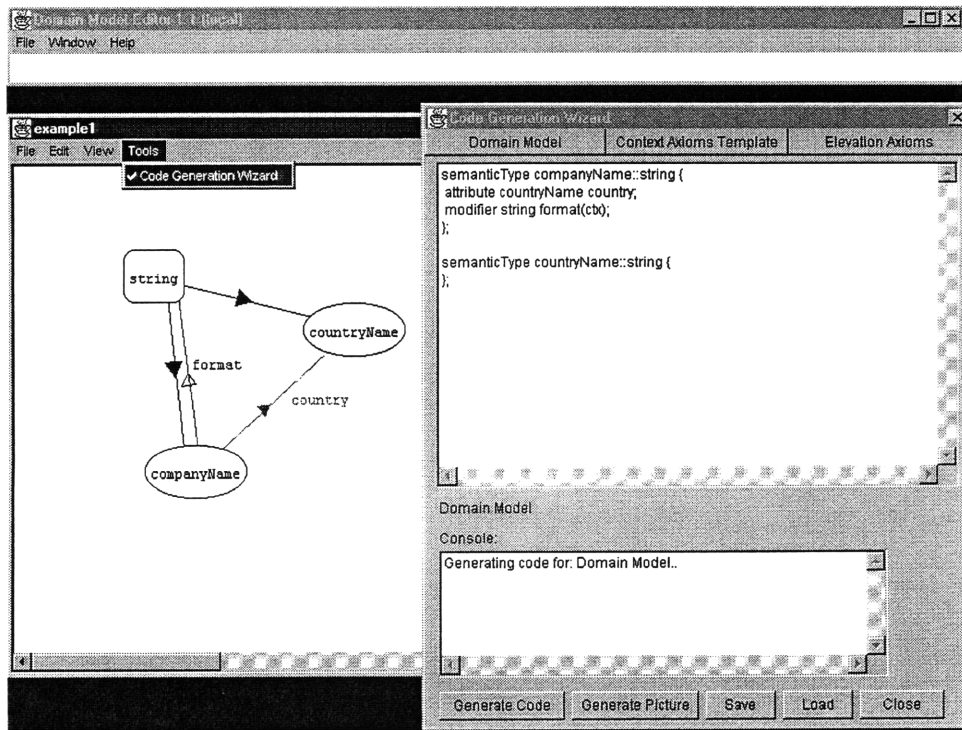


Figure A-6 The Code Generation Wizard Box

After you open the Code Generation Wizard Box, you will see three buttons on the top of the box labeled: “**Domain Model**”, “**Context**”, “**Elevation Axioms**”

**Axioms Template**”, and **“Elevation Axioms.”** These buttons correspond to three different displaying panels within the Code Generation Wizard Box. In the next release version, the Code Generation Wizard module will also include code generation for PROLOG. Currently, it is only compatible with PENNY code

Click on each of the three buttons to display each panel:

### **Domain Model Panel**

Figure A-6 displays the PENNY code that corresponds to the picture in the canvas of the Document Window. The editing area in the Domain Model panel should be empty when you first open the Code Generation Wizard Box. To generate the code of the domain model that you just drew, click on the **“Generate Code”** button at the bottom of the box. The **“Generate Picture”** button next to it is for generating the picture from the code shown in the editing area. It is useful if the user chooses to edit the code instead of editing the picture. The editing area shows the following code generated from the picture:

```
semanticType companyName::string {  
  attribute countryName country;  
  modifier string format(ctx);  
};
```

```
semanticType countryName::string {  
};
```

You can see how the semantic types are defined in the PENNY code shown above. The picture in Figure A-6 shows two semantic types: **“companyName”** and **“countryName”**, both inherited from the system type: **“string”**, which appears after the **“::”** sign. Within each semantic type declaration there exist the modifiers and attributes belonged to it. Notice that **“countryName”** has no modifier or attribute defined. The modifier and attribute links are pointing from **“companyName”**, meaning that they belong to **“companyName”**.

### **Context Axioms Panel**

This is the panel that provides the editing of context axioms — definitions of modifiers defined for a particular context. The context axioms template can be retrieved from the domain model by first clicking the **“Context Axioms Template”** button and then click **“Generate Code.”** The reason to call the panel **“Context Axioms Template”** because the domain model only provides the modifiers information for the set of context axioms. Users have to input the extra conversion functions specific to the context. You can go ahead and try opening the Context Axioms panel, typing in the Context name (name of the source), and hit the **“Generate Code”** button:

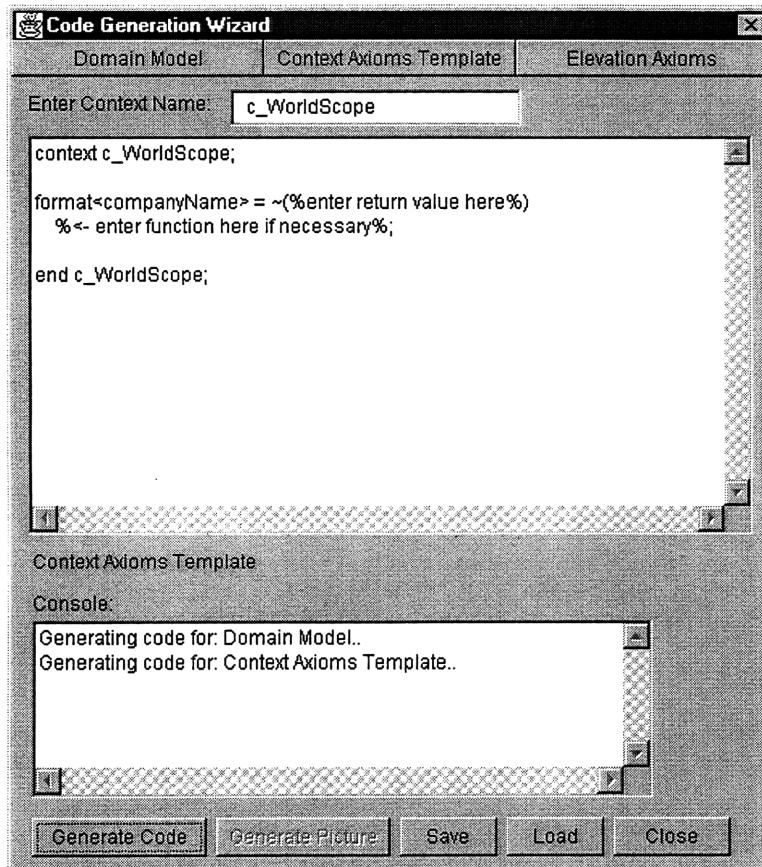


Figure A-7 Generating Context Axioms

The generated code in the editing area shows the declaration of the modifier: “format” of semantic type “companyName”. Notice that the return value and the conversion functions for the modifier are missing because the users have to determine them. In this simple example, the return value is simply a format type named “ws\_name.”:

```
context c_WorldScope;

format < companyName > = ~("ws_name");

end c_WorldScope;
```

The conversion function is omitted because “ws\_name” is a constant string value. In some complicated case in which the return value of, say, *currency*, has to be derived from some other parameters such as *countryIncorp* and *currencyType*, the users have to input the conversion functions. For example:

```
...
currency < companyFinancials > = ~($) < .
  Comp = self.company.
  Country = Comp.countryIncorp,
  CurrencyType = Country.officialCurrency,
```



\$ = CurrencyType.value;

...

The variables: *Comp*, *Country*, and *CurrencyType* are intermediate values that give the final result stored in \$.

### Elevation Axioms Panel

The Elevation Axioms panel provides the interface for mapping a piece of data from the data source to the attributes of the semantic types in the domain model. Defining the set of elevation axioms for a data source requires the users a knowledge of the data to be retrieved. For example, a web site called *Worldscope* provides the following data that the users want to map to the domain model:

<b>Parameter Name</b>	<i>cname</i>	<i>incorp</i>
<b>Description</b>	Name of the company	Name of the country that the company belongs to
<b>Semantic Type Being Mapped To</b>	companyName	exchangeRate

Figure A-8 Assigning data to semantic types

The first row shows two pieces of data, or *entities*, which can be retrieved from Worldscope, the data source. The values are applied to the semantic types that represent them. To assign entities values to the domain model, open the Elevation Axioms panel by clicking “Elevation Axioms” in the Code Generation Wizard Box. Then click on the “Assign Entities” to display the assigning entities section:

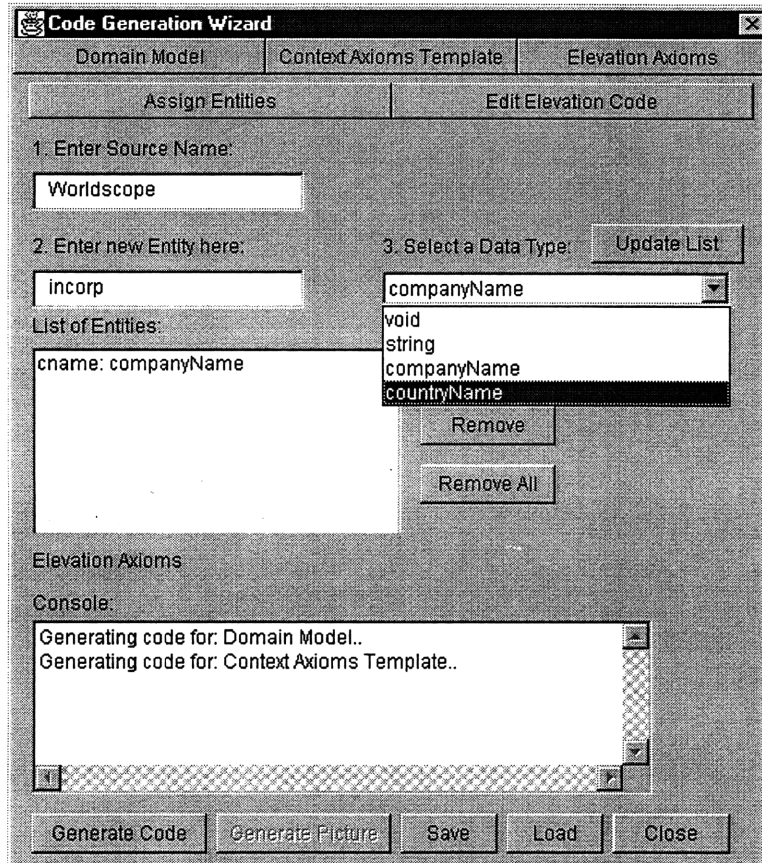


Figure A-9 Assigning Entities to the Domain Model

There are three input fields in the “Assign Entities” section:

1. Enter source name – enter name of the data source here: e.g. *Worldscope*.
2. Enter new entity – type in entity here one at a time, e.g. *cname*, *incorp*, etc.
3. Select data type – from the choice box, select a data type (system or semantic) defined in the domain model.
4. Click on “Add” to add the new entity. It is above the “Remove” button.
5. Repeat steps 2 to 4 for adding a new entity.

After adding all the new entities, click on “Generate Code”. The code will appear in the elevation axioms editing box:

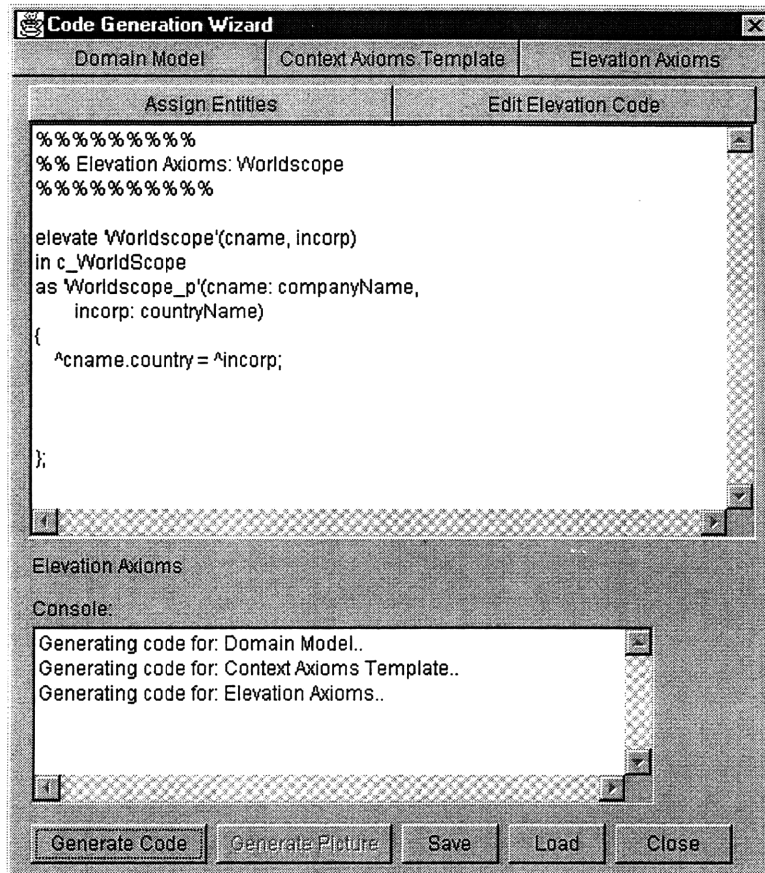


Figure A-10 Generating Elevation Axioms

Let us look into the declaration part generated code:

```
elevate 'Worldscope'(cname, incorp)
in c_WorldScope
as 'Worldscope_p'(cname: companyName,
incorp: countryName)
{
 ^cname.country = ^incorp;
};
```

The elevation set contains two entities: **cname** of type *companyName* and **incorp** of type *countryName*. The label “c\_WorldScope” represents the context name specified in the Context Axiom panel. The body part labeled “Worldscope\_p” assigns the entities values to the related attributes. Back to the Domain Model in Figure A-6, *companyName* has an attribute called “country” whose value is given by “countryName”, to which **incorp** is assigned. The result is shown in the line:

```
^cname.country = ^incorp;
```

The symbol '^' denotes the assignment of a value. Again, this simple example shows the assignment of values that do not require any conversions. A more detailed example will follow in the next chapter.

## Saving and Loading the Documents

The file saving and loading mechanisms are rather complicated because they involve different formats. Also, due to the security feature of web browsers, any Java applet that a browser downloads from a server and running on the client machine is unable to use the save, load, as well as print functions on the client side. The Domain Model Editor program will detect and decide whether the file saving and loading are to be done on a remote or local machine.

### Saving

- a) From menu, choose File → Local or Server → Save As
- b) Select the directory and type the file name in the File Dialog Box.

All the files are in ascii text format. There are two ways to save a domain model:

1. Save as Picture with x, y coordinates for each component.

An example output file has the following format (note: Do not modify any of the code. Only make the change within the Domain Model Editor):

```
%% Nodes %%  
System 0 -1 string (120 94)  
Semantic 1 0 companyName (155 260)  
Semantic 2 0 countryName (291 140)  
  
%% Links %%  
Attribute 0 1 2 country (162 256) (279 150)  
Inheritance 1 0 2 (136 100) (275 135)  
Inheritance 2 0 1 (132 112) (147 250)  
Modifier 3 1 0 format (159 249) (140 105)
```

The first three rows store the values for the three nodes of the Domain Model in Figure A-6, and the rest store the values for links. The first number after "System" and "Semantic" shows the index of the node, the second number the index of its parent, i.e. its supertype. The "-1" means the system node "string" is not assigned to any type as a subtype. The numbers in the brackets show the x, y coordinates of the node.

Similarly, the first number of the link is the link index, the second is the index of the node that the link is pointing away from, and the third is the

index of the node that it is pointing. The rest shows the x,y coordinates of the starting and end points.

## 2. Save as Domain Model in PENNY format

As shown in Figure A-6, the PENNY code in the domain model section describes the same thing as in the picture format, except that it does not store the coordinates. When the program retrieves the domain model picture from the PENNY code, it will generate the coordinates randomly. Therefore, the users have to spend some time arranging the components.

To save the context axioms and elevation axioms code, simply click the “Save” button on the Code Generation Wizard Box. A dialog box will show up. Select the directory and type in the file name. Although not strictly enforced, try using the following extensions:

- Picture: **.txt**
- Domain Model: **.pny**
- Context Axioms: **.ca**
- Elevation Axioms: **.ea**

## Loading

The loading part is also as straight forward except that the users might easily make mistakes by loading the wrong file into a section of the Code Generation Wizard Box. The console in Wizard Box will display a warning line if it recognizes a wrong file format. For example, in the Domain Model panel of the Wizard, if you click “Load” and choose the a file, the program will try to detect the word “*semanticType*” that appears in every PENNY Domain Model file. If not found, the console will display, “Error: File might be in wrong format.”

### 1. Loading the picture

- a) From menu, choose File → Local or Server → Open → Picture
- b) Select file from the File Dialog Box.

### 2. Loading the code

- a) Domain Model in PENNY, from menu:

Choose File → Local or Server → Open → PENNY code

or from Tools → Code Generation Wizard:

Click on “Domain Model” button to display panel, click “Load.”

- b) Context Axioms File

From Tools → Code Generation Wizard, click on “Context Axioms” button to display panel, then click “Load” and choose the file.

c) Elevation Axioms

From Code Generation Wizard, click on “Elevation Axioms” button to display panel, then click “Load” and choose the file.

## A2. Functional Specifications & Class Descriptions

---

### IN THIS SECTION

- Class Hierarchy
- Functional Specifications of Each Class and Description of All Functions

## Introduction

The previous chapter gives a discussion on the architectural design of the Domain Model Editor System. This chapter looks more deeply into each class and describes the functionality. The first section lists out the hierarchy of the components in the application. Each of the class inherits from the standard Java 1.1 API package developed by SunSoft. The next section describes the functionality of each method of each class. Any Java compiler obtained from the Java Development Kit Version 1.1.x should be able to compile all the code. Any web browser that has Java Virtual Machine and supports Java applets 1.1 can run the entire application.

## Class Hierarchy

Key: ◆ — User defined classes for this Application  
● — Classes developed in Java Development Kit 1.1.x

### 1. User Interface Classes

- class java.lang.Object
  - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - class java.awt.Container
      - class java.awt.Panel
        - class java.applet.Applet
          - ◆ class **MainApplet**
      - class java.awt.Window
        - class java.awt.Frame (implements java.awt.MenuContainer)
          - ◆ class **MainWindow** (implements java.awt.event.ActionListener)
          - ◆ class **DocumentWindow** (implements java.awt.event.ActionListener, java.awt.event.ItemListener)
    - class java.awt.Dialog
      - ◆ class **CodeDialog** (implements java.awt.event.ActionListener)
      - ◆ class **PaletteBox** (implements java.awt.event.ActionListener)
      - ◆ class **YesNoDialog** (implements java.awt.event.ActionListener)
      - ◆ class **MessageDialog** (implements java.awt.event.ActionListener)



- class java.awt.Container
  - class java.awt.Canvas
    - ◆ class **MainCanvas** (implements java.awt.event.MouseListener, java.awt.event.MouseMotionListener, java.awt.event.KeyListener)
    - ◆ class **MultiLineLabel**

## 2. Domain Model classes

- ◆ class **DomainModel**
- ◆ interface **Node**
- ◆ class **Systemnode** (implements **Node**)
- ◆ class **Semanticnode** (implements **Node**)
- ◆ interface **Link**
- ◆ class **Inheritance** (implements **Link**)
- ◆ class **Modifier** (implements **Link**)
- ◆ class **Attribute** (implements **Link**)
  
- ◆ class **Arrow**

## 3. Server Application Classes

- ◆ interface **FileIOServer** (extends java.rmi.Remote)
- class java.rmi.server.RemoteObject (implements java.rmi.Remote, java.io.Serializable)
  - class java.rmi.server.RemoteServer
    - class java.rmi.server.UnicastRemoteObject
      - ◆ class **FileIOServerImpl** (implements FileIOServer)

## Functional Specifications of Each Class and Description of All Functions

**Class Library Functions Descriptions Available online:**

Please go to the URL at

<http://soloaudio.mit.edu/java/domain11/docs/tree.html>

for the full class library functions of the Domain Model Editor. The release version of the application and documentation will be at our group's main web site in June, 1998 at:

<http://context.mit.edu/~coin/>