Tools for Virtual Merging of Ecoin Applications and Guidelines for Building Ecoin Ontologies

Faisal R. Anwar

Composite Information Systems Laboratory (CISL)
Sloan School of Management
Massachusetts Institute of Technology
Cambridge, MA 02142

Tools for Virtual Merging of Ecoin Applications and

Guidelines for Building Ontologies

By

Faisal R. Anwar

Submitted to the Department of Electrical Engineering and Computer Science

May 20, 2004

in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

The Ecoin system is used to perform context mediation among heterogeneous data sources. Ecoin applications present a fundamental module in the Ecoin system that are used to describe a domain of interest and to perform context mediation on data from that domain. We describe the CLAMP tool, which facilitates merging individual Ecoin applications so that the context mediation functionality of two domains of interest are combined in one single application. We also demonstrate some principles for building one of the most critical parts of an Ecoin application – the ontology – and build a sample ontology step-by-step using these principles.

Thesis Supervisor: Stuart E. Madnick
Title: John Norris Maguire Professor of Information Technologies,
MIT Sloan School of Management and
Professor of Engineering Systems,
MIT School of Engineering

Table of Contents

Table of Figures

Chapter 1. Introduction

We live in an increasingly information-dependant world, and much of the information that abounds has with it some inherent assumptions. Stock prices that we read from our favorite Internet sources have basic assumptions about the currency and format that the price is given in. When we retrieve data about times and dates of events, those values are passed in a specific format that we (as humans with naturally built in context mediators) can often interpret without much difficulty. As these examples suggest, all data around us has specific assumptions behind it that dictate how people will interpret the meaning of that data.

Although we can often interpret the meaning of data with ease, there are many cases where we humans err in the assumptions we make about the meaning of information. These misjudgments come about as a result of a lack of complete knowledge about the specific assumptions made about data – a lack of knowledge about what *context* data is to be interpreted within. As a result, we may make errors in judging how much money a value actually represents because we don't know the units or currency behind that value. We may be confused about what time "3:43 PM" represents because we have limited information about the time zone associated with this value.

In addition to the difficulties that we as humans have in interpreting data, software agents that must automatically process data also don't have the capability to interpret data with reference to different contextual assumptions. One field of need for the context mediation is the Semantic Web, a vision of the evolution of the current World Wide Web [1]. In the Semantic Web environment, it is hypothesized that software agents will be able to retrieve, process, and act upon data in an intelligent manner. These agents can't communicate and successfully process information without interpreting that information in the correct manner. As a result, the Semantic Web provides just one of the many areas of need for context mediation.

These basic problems provide the *raison d'être* for the Ecoin system. The Ecoin system performs the automatic context processing that most software agents require, and that we humans will often require, in order to interpret information correctly. Ecoin sits between data and users of that data and converts the data as appropriate in order to align the contextual assumptions implicit in the data with the assumptions expected by the users of Ecoin [4].

This thesis continues work on the Ecoin system with tools that extend its capabilities and guidelines meant to aid those individuals or entities that would like to mediate data using the Ecoin system. Eventually, we will go in to detail about the contributions to existing context mediation research that this thesis makes. However, we start with some concrete examples that will help frame the need for Ecoin as well as its use and development.

1.1. Example Scenarios Discussed

This thesis will consider three basic problems when discussing the Ecoin system in general as well as the contributions to the systems that this thesis makes. The first two examples are borrowed from Kaleem's thesis [3], and provide a vehicle to help us discuss a tool for merging *Ecoin applications*. The notion of an Ecoin application shall be discussed in more detail later, but put simply, it refers to the data sources and underlying information about the data sources that is necessary to perform context mediation. Ecoin applications are usually created on a per domain basis: each Ecoin application describes and mediates a specific domain of knowledge – such as the three discussed below.

In addition to Kaleem's Airfare and Car Rental examples, I will also present a scenario where one seeks to mediate data about companies – stock information, company financial reports, etc. This will provide a vehicle for discussing the guidelines for building Ecoin ontologies – an essential part of any Ecoin application.

1.1.1. Airfare Aggregation

Many websites exist today to provide purchasing and availability information for airline tickets. Some popular sites with this information include Orbitz and Travelocity, shown in figure 1.1. Typically, a user enters information about the time and destination of his trip, and the sites spit out a listing of the available flights, their exact times of departure and arrival, and purchasing information. Figure 1.1 displays information that is returned when you query the websites for flights between Boston and New York, with a departure date of 6/1/2004 and a return date of 6/15/2004.

Figure 1-1: Sample data sources for the airfare aggregation domain.

So why would we need the Ecoin system to perform context mediation on this type of data? There are many inherent assumptions that exist for the data returned from the various airline ticket websites – especially with regards to the prices that a user sees [3]. Some specific points of *contextual conflict* (conflicting assumptions about the meaning of data) include:

- The prices quoted by the online services may not be in the same currency as that assumed by the user. What if all of the sources quote prices in dollars, but our user is someone planning a trip around the U.S. when he visits the country from the U.K. This person would assume that prices are in dollars? What if our user is indeed American, but he queries an international flight that is quoted in a foreign currency?

- Airline prices are also known to include many additional charges that are not displayed in the basic query. These charges include service fees that the website adds on for finding the tickets for the user, additional charges for printing out paper tickets, and additional charges for visa fees if they apply in an international flight. Users may want to see the prices with all of these additional charges accounted for.

- For longer flights, there may be a stopover in between the departure and arrival points. Some users may want the flight times reported to account for these stopover times, or to at least view all flights from the different websites with flight times included or excluded.

These are just some of the contextual issues that may come up when querying data in the airline aggregation domain. For this and the next two domains, Ecoin applications are created to mediate these contextual conflicts and provide a view of the data consistent with the assumptions that users have made about the data.

1.1.2.   Car Rental

The next domain we consider that requires context mediation is online services for renting cars. Similar to the way the airfare websites require users to enter information and give back available tickets, car rental websites take user input about the date and location for their rental, and spit out a listing of available rentals from different providers. Expedia (http://www.expedia.com) and Yahoocar (http://travel.yahoo.com) are examples of such online service.

This domain also has several context issues. Among them are:

- Different sites may report the rate of rental in different ways. One site may report the cost of rental on a per-day basis, while another site reports the cost for the entire period. Furthermore, different users may want to see the cost of rental differently – some as the aggregate cost for their entire trip, and others as the cost per day or per week.

- The different car rental data sources may differ on what is included in the prices they report. Some may include the cost of insurance and taxes, while others may not.

- The format that location input is provided may differ across sites. Some sites may require the input to correspond to a local airport, while others may require city or location codes.

The Ecoin system can be (and has been) used to mediate such contextual conflicts. The goal of this thesis is not to repeat the work of Kaleem [3] in building the infrastructure for mediating the contexts that exist in the airfare and car rental domains. For detailed information on how this is done, refer to Kaleem [3]. I will use these examples to demonstrate the functionality of an extension to Ecoin – the CLAMP tool for merging individual applications (discussed in later chapters).

1.1.3. Company Information

The final domain that is of interest for this thesis is the public corporation domain. Let's assume you are CEO of a hypothetical investment company, Foo Holdings Incorporated (FH Inc.), which buys company stocks all over the world. Two of your most important offices happen to be in New York and London.

In order to make informed investment decisions, your company needs a large collection of information – the financial details of a company, how its stock has been performing, etc. Several online resources exist to provide you with the data that is needed, including Yahoo's finance website (http://finance.yahoo.com), and Bloomberg online (www.bloomberg.com). Both sites provide an abundance of information that various analysts use to make decisions on where to invest the billions that the company is endowed with. Figure 1-2 on the next page shows some of the data that one might access from one of these sources.

In the business world, however, time is of the essence. Analysts can't be spending their valuable time trying to figure out whether a quote is in one currency or another as they look up information from stock markets across the world. An incorrect guess on the units used to display the market capitalization of Microsoft can cost FH Inc. a lot of money. Company information has several such context issues that must be resolved. Among these are:

1) Currency differences in the way monetary data (such as stock prices, revenues, etc.) is reported.
2) Differences in the scale factors used to report monetary values.
3) Differences in how the ticker symbols for companies are represented. Some sites attach a suffix indicating the stock market that the stock is being quoted from.

Furthermore, accessing web sources can often be slower than having a proprietary data source that provides you with all of the information you need. As a result, FH Inc. is thinking of making a new internal source of company information available in the near future in addition to the online web sources.

YAHOO! FINANCE

MICROSOFT CP (NasdaqNM:MSFT) Quote data by Reuters ___ Edit

After Hours (RTM/ECN): 26.13 0.00 (0.00%)

Last Trade:	**26.13**	Day's Range:	25.96 - 26.75
Trade Time:	Apr 30	52wk Range:	23.5999 - 30.00
Change:	↓ 0.35 (1.32%)	Volume:	66,175,600
Prev Close:	26.48	Avg Vol (3m):	65,675,500
Open:	26.59	Market Cap:	281.60B
Bid:	8.43 x 100	P/E (ttm):	38.43
Ask:	43.62 x 100	EPS (ttm):	0.68
1y Target Est:	34.04	Div & Yield:	0.16 (0.60%)

MSFT 30-Apr @ 4:00pm (C)Yahoo!
26.8 26.6 26.4 26.2 26.0 25.8
10am 12pm 2pm 4pm
1d 5d 3m 6m 1y 2y 5y max
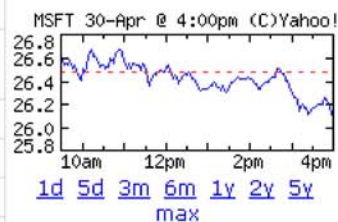
Annual Report for MSFT

FINANCIAL HIGHLIGHTS

Fiscal Year

Fiscal Year Ends:	30-June
Most Recent Quarter (mrq):	31-Mar-04

Profitability

Profit Margin (ttm):	20.78%
Operating Margin (ttm):	22.39%

Management Effectiveness

Return on Assets (ttm):	8.89%
Return on Equity (ttm):	11.22%

Income Statement

Revenue (ttm):	35.61B
Revenue Per Share (ttm):	3.274
Revenue Growth (lfy)[3]:	13.50%
Gross Profit (ttm)[2]:	26.50B
EBITDA (ttm):	9.36B
Net Income Avl to Common (ttm):	7.40B
Diluted EPS (ttm):	0.68
Earnings Growth (lfy)[3]:	27.60%

Balance Sheet

Total Cash (mrq):	56.41B
Total Cash Per Share (mrq):	5.23
Total Debt (mrq)[2]:	0
Total Debt/Equity (mrq):	0
Current Ratio (mrq):	4.083
Book Value Per Share (mrq):	6.549

Cash Flow Statement

From Operations (ttm)[3]:	14.86B
Free Cashflow (ttm)[3]:	14.06B

Microsoft Corp
One Microsoft Way
Redmond, WA 98052
Phone: (425) 882-8080
Fax: (425) 936-7329
Email: msft@microsoft.com
Web Site: http://www.microsoft.com/

REUTERS ABRIDGED BUSINESS SUMMARY

Microsoft Corporation develops, manufactures, licenses and supports a wide range of software products for various computing devices. The Company's software products include scalable operating systems for servers, personal computers (PCs) and intelligent devices; server applications for client/server environments; information worker productivity applications; business solutions applications, and software development tools. Microsoft provides consulting services and product support services, and it trains and certifies system integrators and developers. The Company sells the Xbox video game console, along with games and peripherals. Its online businesses include the MSN subscription and the MSN network of Internet products and services. The Company's seven product segments are: Client, Server and Tools, Information Worker, Microsoft Business Solutions, MSN, Mobile and Embedded Devices and Home and Entertainment.

Figure 1-2: A view of company information data that users may want to access when making investment decisions.

Such circumstances necessitate the use of the Ecoin system. The above scenario describes a likely problem that individuals or firms may be faced with and that requires them to use automated context mediation. Later in this thesis, I shall use the above scenario to describe the situations where Ecoin can be of help and to provide guidance on building Ecoin ontologies needed to perform context mediation.

1.2.   Previous Work on Ecoin

Before we discuss the specific goals and contributions of this project, it is instructive to give a brief history of related work that describes the Ecoin system. The work that has preceded this thesis provides crucial details that the reader may want to refer to in order to completely grasp the concepts discussed in later chapters.

Firat [4] provides the theoretical background and foundational modules that make up the Ecoin system. He describes Chameleon, a tool for wrapping websites such as the ones described above so that they can be accessed as if they were a database. Once a website is wrapped, it can be queried using Structured Query Language, with Chameleon returning the data in tabular or XML format [3]. Firat also defines the notion of *semantic heterogeneity*, a term that essentially refers to the contextual diversity of data that we've discussed above.

Firat also describes in detail the strategy of the Ecoin system for mediating contextual conflicts in various domains. He documents how the theoretical goals of Ecoin are then implemented at the system level using a reasoning engine in prolog as the core part of the mediation system. Finally, Firat gives the theoretical basis for one task that is a key part of this thesis: merging Ecoin applications. Firat describes in detail what exactly application merging refers to in the context of Ecoin, and how this process differs from related literature on ontology merging [4].

Lee's work describes the Ecoin implementation in some detail and presents a new infrastructure that facilitates Ecoin application creation and management [2]. This tool, known as the Application Editor, provides a web-based interface for creating and modifying Ecoin applications.

Finally, Kaleem implements Firat's work on merging and describes the manual creation of a merged application using the airfare and car rental applications described above. Furthermore, he discusses an initial design for CLAMP – a tool that facilitates Ecoin application merging [3].

1.3.   Goals and Contributions of this Thesis

This thesis extends the Ecoin system to facilitate application merging and also provides guidance on what an Ecoin application is and how one goes about creating the ontology for an Ecoin application.

1.3.1. Ecoin Application Merging and CLAMP

The first major contribution of this thesis is the exposition of the merging process using a newly developed tool called the CLAMP tool. CLAMP stands for Context Linking and Merging Process, and refers to a process where two complete applications representing different domains (such as airfare and car rental) can be merged in to one single Ecoin application. I extend the work done by Kaleem [3], and Firat [4] in the following ways:

- The objectives of the CLAMP tool in facilitating merging are discussed. In addition to its ultimate goal of facilitating the merging process, this thesis discusses the important design issues that must be considered when designing and implementing a tool to facilitate merging.

- There is a detailed discussion of the design and implementation of the CLAMP tool. I describe how Lee's Application Editor is extended to support the necessary infrastructure for merging applications. I also implement algorithms that give merged Ecoin applications the theoretical properties that help to provide a consistent view of what an Ecoin application is to the Ecoin system as well as to those using the system.

- Finally, there is a detailed discussion on the user interface of the CLAMP tool. This provides explanation of how one would go about merging two applications using the tool.

1.3.2. Other Goals and Contributions of This Thesis

The other major category of contributions fall under the notion of making Ecoin applications an easier and clearer process for users. There has been some discussion by Firat [3] of what comprises an Ecoin application, but this is again given in a manner separated from the actual implementation of the system. Kaleem provides a methodology for creating Ecoin applications, but questions still exist as to how the Ecoin ontology – one of the crucial parts of an Ecoin application – is built. This thesis addresses these issues through the following objectives:

- A discussion of what information comprises an Ecoin application. This discussion helps guide the discussion of application merging and also provides a bigger picture of the role of the Ecoin ontology in facilitating context mediation for a domain of interest.

- The various scenarios that individuals or firms may encounter where Ecoin can serve an important purpose are stated.

- There is also discussion on how one can go about creating the Ecoin ontology. Much literature exists providing guidance on building ontologies, but from purely the perspective of representing a domain of knowledge. The

use of Ecoin ontologies is specific to the task of mediating contextual conflicts. I provide principles and a methodology for building ontologies for use in the Ecoin system for context mediation.

By understanding the notion of an Ecoin application and using the guidance provided on creating ontologies, users in a variety of communities can easily identify the situations where Ecoin will be useful and apply the system appropriately to serve their needs.

1.4.   Thesis Agenda

With the basic objectives for this thesis set, I now proffer the agenda for the rest of the thesis in achieving the goals stated. Chapter 2, entitled "Background Information", provides a basic overview of the key Ecoin concepts that will be essential in understanding later discussions about the CLAMP merging tool and the guidelines for merging Ecoin ontologies. If the reader is curious for more details than those provided in Chapter 2, he may refer to the works of Firat, Lee, and Kaleem for detailed discussion on the Ecoin System, Metadata Management in Ecoin, and Virtual Merging of Ecoin Applications, respectively.

Chapter 3, "CLAMP Design and Implementation", provides a detailed discussion on the CLAMP tool – its goals and objectives in facilitating the merging process, its design, its place in the overall Ecoin system, and any other implementation issues that may be pertinent. Chapter 4 then provides a walkthrough tutorial explanation of the CLAMP user interface, using Kaleem's airfare and car rental applications to demonstrate the capabilities of CLAMP. Parties that are interested in merging Ecoin applications with CLAMP are encouraged to look through this chapter for guidance.

With a complete understanding of the Ecoin system, including the extensions to support merging, we can then move to a discussion on how ideally to use the Ecoin system to suit the needs of user communities. Chapter 5, "Guidelines for Designing Ecoin Ontologies", provides guidance on scenarios where the Ecoin context mediation system can be of use, as well as principles and methods for creating Ecoin ontologies that are at the center of any Ecoin application.

Finally, chapter 6 concludes the discussion of this thesis, provides an enumeration of the objectives achieved, and gives summary of the key points from previous chapters.

Chapter 2. Background on Ecoin and Merging

This chapter introduces some of the basic concepts underlying the Ecoin system and the process of merging Ecoin applications. Much of the terminology that is crucial to understanding the discussion of CLAMP and the guidelines for building Ecoin Applications is introduced here. This chapter is not meant to be an exhaustive documentation of the Ecoin system, but rather a brief review of the major underlying theory and infrastructure that will help give a point of reference for later chapters. For a more detailed discussion of the Ecoin system, the reader may refer to Lee [2] for background of the implementation of the system. Firat [4] has a discussion of the theoretical foundations of the context mediation using the Ecoin approach, as well as a discussion on virtual merging of Ecoin ontologies. Kaleem [3] also discusses ontology merging with the Ecoin system in detail.

2.1.   Introduction to the Ecoin System

The Ecoin system performs context mediation between heterogeneous data sources. Data can be obtained from a variety of data sources (ranging from databases to semi-structured data on websites). Data from several different sources may mean the same thing semantically, but ambiguities may exist in interpreting this data consistently because it is represented in a variety of different ways depending upon the originating data source. Other systems already exist to deal with this problem of semantic conflict, but they require potential conflicts to be identified and resolved *a priori* to the application of the system to a particular set of data. Ecoin, on the other hand, requires only that the user specify how data are to be interpreted in different contexts (corresponding to different sources and users of data) and how conflicts are resolved *if* they come up.

Let's look at an example to clarify the problem that Ecoin attempts to solve. Below is a snapshot of stock quotes from two different websites for stocks in the United States (figure 2-1) and Great Britian (figure 2-2). This information is a simplification of the data I introduced in Chapter 1 regarding the domain of company information. We shall use these figures to guide some of our background discussion on the Ecoin system.
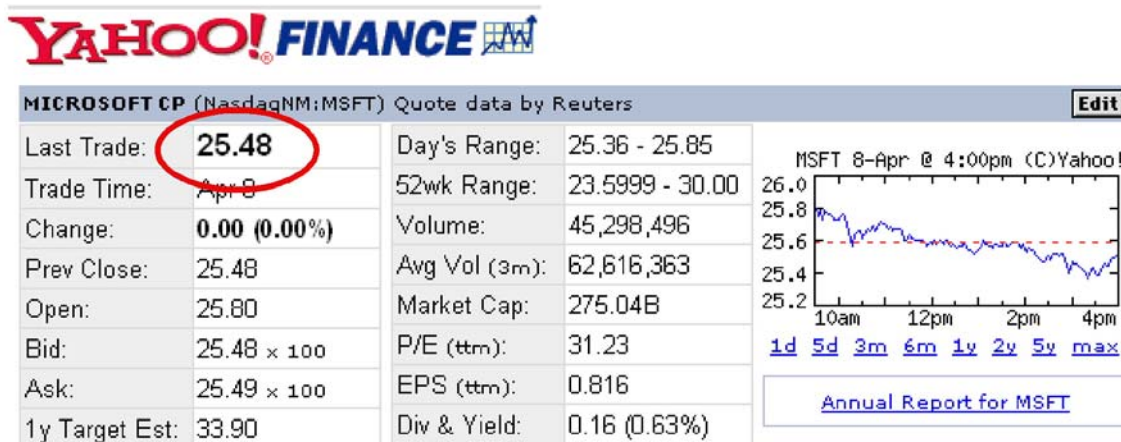
Figure 2-1: Microsoft Stock Quote from Yahoo Finance (NASDAQ Stock Exchange, United States)

Figure 2-2: British Airways stock quote from Bloomberg (London Stock Exchange, Great Britain).

Figures 2-1 and 2-2 are both examples of semi-structured data sources that we can treat as database tables once we wrap the sites using a tool such as the Chameleon web wrapper [4].

The circled items in the two figures theoretically represent the same thing – the price of the respective stocks displayed by the website. However, the data from the quote from Microsoft (a company listed under an American stock market) is in dollars, whereas the currency for the quote on British Airways from Bloomberg.com is in British pence (which is 1/100 of a Pound). There is thus a difference in the currency that the monetary values are represented in, as well as the scale factor with respect to the base currency (dollars or pounds). In Ecoin terminology, these differences would be called contextual conflicts, and the Ecoin system exists to seamlessly mediate between such conflicts. An Ecoin-mediated query retrieving the stock data from these data sources would automatically convert the stock prices to dollars, pounds, or pence depending on what currency and scale factor the user is assuming in his context. The Ecoin system can mediate between contextual conflicts that include differences how data is represented as well as differences in how data is defined.

As a note on terminology, the word *domain* or *community of interest* that Ecoin is being used for is used to describe a particular class of data sources. For example, the

examples above can be considered in the domain or community of interest that has to do with stock information for companies. All data sources that provide such information about company stock prices can be considered a part of this domain.

2.2. Ecoin System Overview

Ecoin performs context mediation through a system design outlined in figure 2-3. The system is designed so that all potential types of contextual conflicts are specified *a priori* to use of the system in fetching queries from a particular domain of data sources. Each module within this overall system is described below with respect to its functionality and role in processing a user's query.
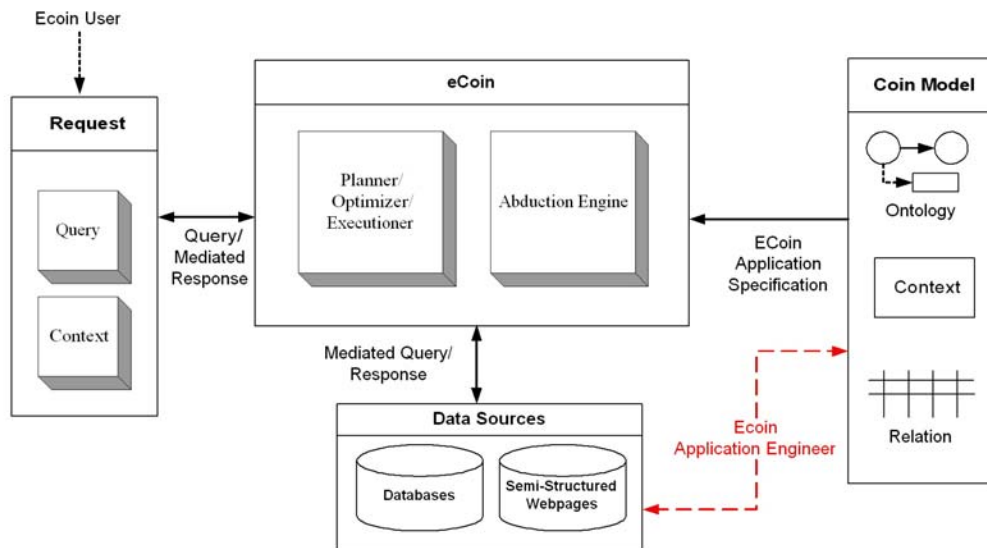
Figure 2-3: The Ecoin system architecture.

2.2.1. The Modules that Make Up the Ecoin System

At the heart of the Ecoin system is the Abduction Engine and Planner/Optimizer/Executioner (POE). These modules are ultimately responsible for taking a user query, modifying it to resolve any contextual conflicts, and running the query on the data sources. The Abduction Engine uses information about the context of the query (i.e. the context of the user submitting the query), as well as pre-specified information about the domain of the query and any contextual conflicts to modify the query so that it resolves any context issues prior to sending results back to the user. The POE module is responsible for obtaining the necessary data from data sources to execute an input query.

A request to Ecoin consists of the query that we've just mentioned as well as information about the context that the user is assuming when submitting the query. In the Ecoin system, the notion of context is tied to the data sources being queried as well as the user submitting the query. The context for the data sources specifies the underlying

assumptions about how the data is defined and represented in those sources, while the context information about a user informs the system the assumptions that the user is making about the data. Context mediation of the request is thus performed such that data in the data sources' contexts are converted to values that are consistent with the request context.

The data sources that a query ultimately seeks information from can be normal database relations, or they can be semi-structured data sources. An example of this would be the stock price web pages that were shown in figures 2-1 and 2-2. Using a web wrapper such as Chameleon [4], these pages can be presented to the Ecoin system as database relations and queried as such.

The Coin Model consists of the ontology, context information, and relation information necessary for the Abduction Engine to create mediated queries. The ontology represents a conceptualization of a particular domain of knowledge, including the objects and relationships that exist within that domain. The context part of the Coin Model includes information about which contexts exist (based on the data sources and users that will be using those data sources). Finally, relations describe the database tables that will be queried using the ontology and context information provided in the Coin Model.

2.2.2. The Ecoin User and Application Engineer Roles

As figure 2-3 shows, there are two key roles that Ecoin users play during the operation of the system. These roles will be termed *Ecoin User* (or user for short) and the *Ecoin Application Engineer* (or application engineer).

The Ecoin User is the client that is submitting queries to the Ecoin system. It is this individual that the Ecoin system ultimately seeks to serve. The goal is to provide the Ecoin User data that is context mediated when he submits a query. Each Ecoin user has a context associated with him that defines all of the assumptions the user makes about the data that he is querying. Relating this back to the example of quoting stock prices, we may have two different users querying the "database" of quotes – one user in the U.S. and another in the U.K. Each of these users must be identified, and a context assigned to them. This context could possibly be the same context as one of the data sources, in which case a new user context isn't necessary. However, if a user doesn't have the exact same implicit assumptions about the data as one of the data sources, then he needs to be assigned a new context and his unique assumptions must be codified through the appropriate mechanism in Ecoin.

An Ecoin Application Engineer, on the other hand, is not responsible for submitting the queries, but rather for designing an Ecoin Application that is then used by the core Ecoin system to mediate queries to a particular domain of data sources. In section 3, I discuss in detail what an Ecoin Application consists of. The Ecoin Application Engineer is responsible for defining this application. This responsibility includes collecting the set of data sources (and wrapping any web sources), defining the domain model (i.e. ontology) for these data sources, incorporating contextual

assumptions in to the application, and defining other relationships necessary for the core Ecoin system to be able to successfully context mediate a query.

Of course, these roles don't necessarily have to be performed by completely different people. In fact, most of the time the person or organization submitting queries will also be engineering the Ecoin Application. However, the distinction is important because there is a clear separation of responsibilities between the two roles and it is quite possible that different individuals may sometimes perform them.

2.3. The Ecoin Application

The Ecoin Application is one part of the overall Ecoin system that performs context mediation of queries. It is especially important to draw a boundary defining this particular component of the system as our discussions in later chapters regarding merging and designing applications will center around this fundamental component. Figure 2-4 will be a point of reference for the discussion about the Ecoin Application.
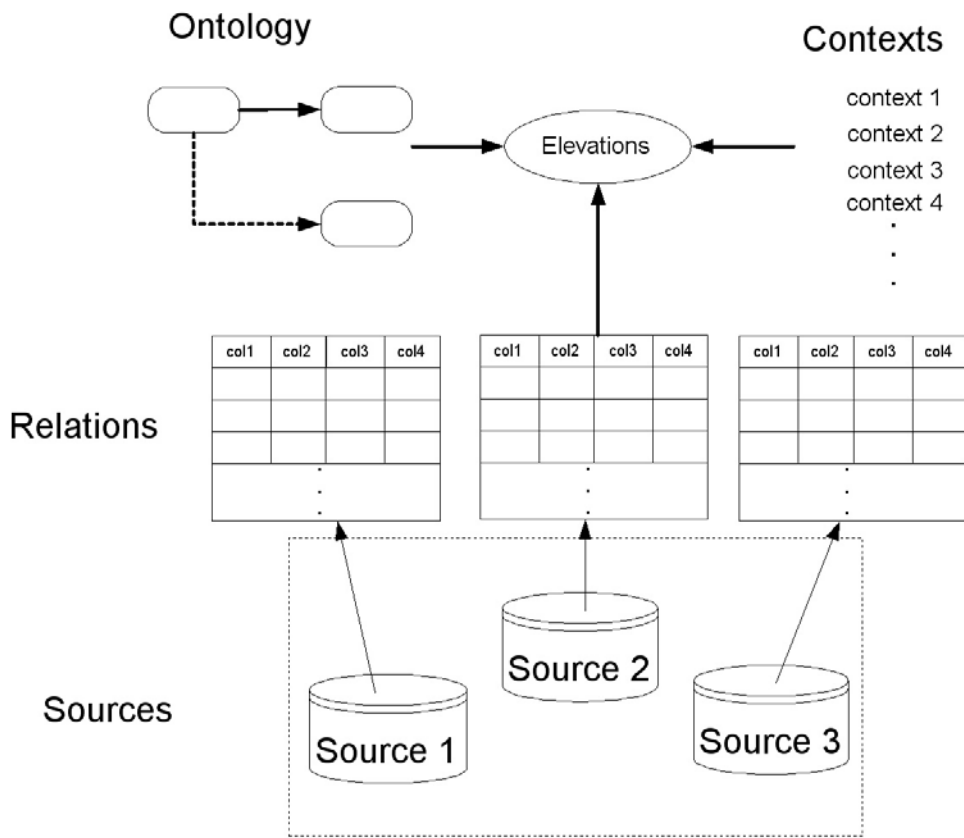
Figure 2-4: The components of an Ecoin application.

The Ecoin Application is the customizable part of the Ecoin system. As discussed in section 2, the core of the Ecoin system consists of the POE and the Abduction Engine. These are static and do not change with different domains of use for the system. The

Ecoin Application, however, must be defined for each specific domain that requires context mediation. Thus, if you have one set of data sources that you will query to look up stock quotes, then an Ecoin Application consisting of the ontology, context, source, and elevation information for that domain must be defined. If a new class of sources and users (lets say people looking up Car Rental information) are being considered, then a separate Ecoin application must be defined for this new domain of use. We shall come back to the issue of merging two Ecoin Applications in the next section. In that case, instead of defining completely independent applications for different sets of data sources, we merge applications so that we can achieve specific advantages that come from including many sources together in one application.

Each Ecoin Application is composed of 6 main elements that an Ecoin Application Engineer is responsible for defining. Let's go through each of these parts of the application in sequence. I shall use the stock quote example to concretize the discussion of each point.

1) **Sources** – The sources represent the different data sources that require context mediation. These can be database tables or websites with semi-structured data that is wrapped using a tool such as Chameleon [4]. For the stock quote application, the data sources are websites with semi-structured data for stock prices, shown in figure 2-5.

21
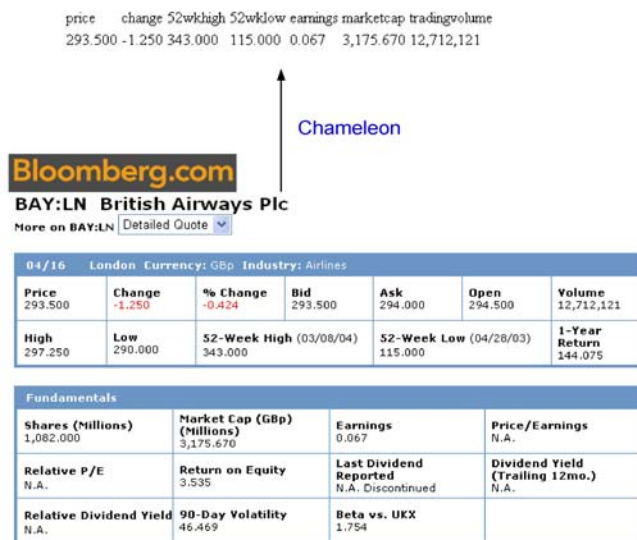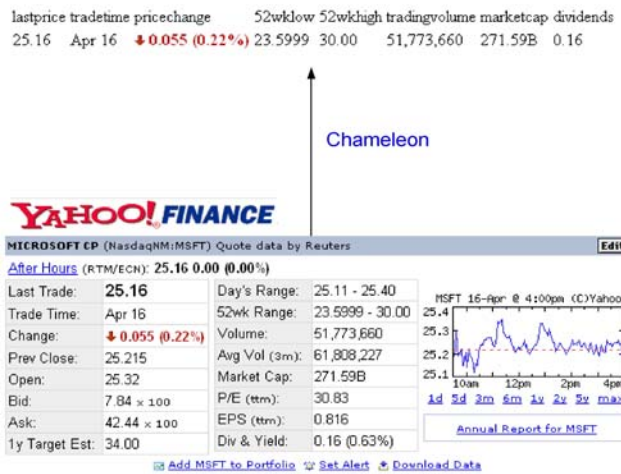
Figure 2-5: Data sources for an Ecoin application. With semi-structured web sites, the sources are wrapped with Chameleon and are then presented as database tables.

2) **Relations** – Defining the relations that exist within an application is simply a matter of naming and describing the sources that the Ecoin system will access. In the stock example, this involves naming each of the two relations – "yahoo_us" for the data obtained from Yahoo! Finance, and "bloomberg_uk" for the London Stock Exchange data obtained from Bloomberg.com.

3) **Contexts** – Identifying the contexts for an Ecoin Application involves identifying the different sources and users for the application and enumerating the different context assumptions that exist for each of these entities. Each user or source that has a different set of assumptions for data is assigned a unique context. This context must then be described by assigning specific modifier values in the application ontology. It is important for the reader to understand that a context describes a unique assignment of modifier values in the underlying ontology. The assumptions that are inherent for each context are made explicit through these modifier value assignments. In our example, we have four contexts of interest. "Bloomberg_uk" and "yahoo_us" are the names of contexts corresponding to the source relations of the same name. "office_us" and "office_uk" are contexts for two different sets of users – one located in the U.S. and one located in the U.K. The purpose of having these additional contexts is that, even if the U.S. and U.K. offices share common context assumptions with yahoo_us and Bloomberg_uk, respectively, there may be other contextual assumptions (such as the format of data) that they do not share with those contexts. Thus, it is necessary to create these separate user contexts to allow users to perform context mediation, even on data from the same country.

4) **Ontology** – The ontology represents the objects and relationships that exist for a particular domain of sources. An ontology consists of semantic types that represent the objects, attributes that represent the relationships, and modifiers which are specialized attributes that codify contextual assumptions behind different objects within the application. Ecoin ontologies are described in detail by Lee [2] and Firat [4]. Below is the Ecoin ontology for the stock quote example.
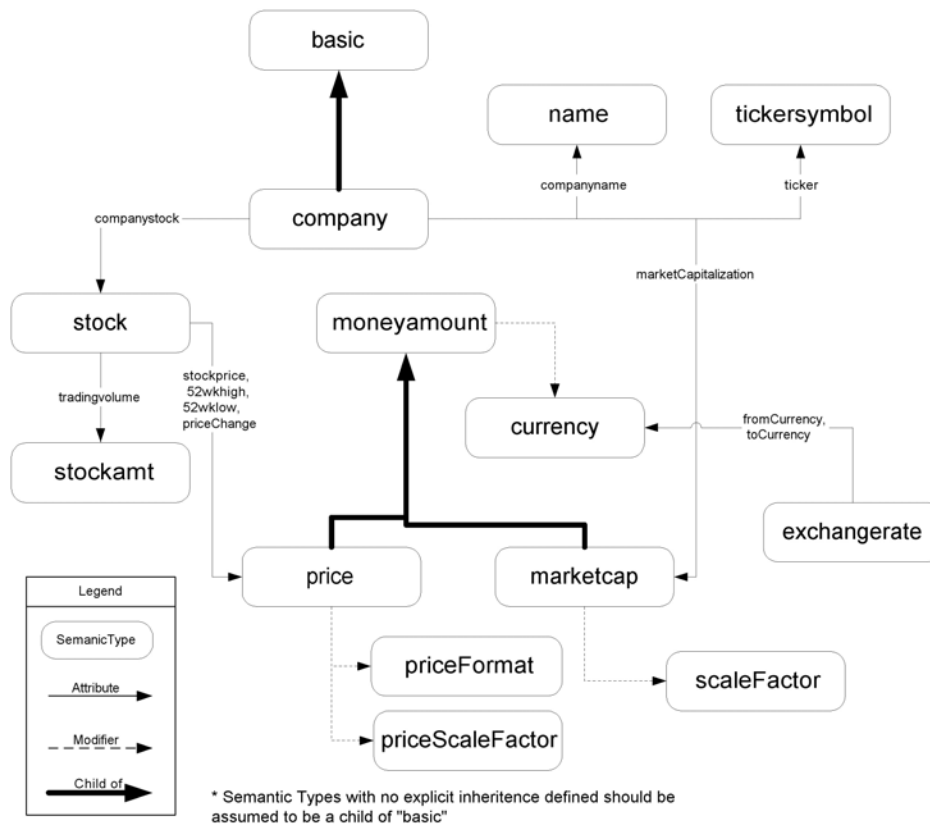
Figure 2-6: The ontology for the stock quote application.

5) **Elevations** – The elevations for an application define a mapping between columns in the source relations, contexts, and semantic types. They essentially indicate which data columns from the sources correspond to which semantic types, with the context also specified in the mapping. In the stock quote application, for example, the elevation (Price, yahoo_us.price, "yahoo_us") may be defined to specify that the column price from the yahoo_us relation (corresponding to data from Yahoo! Finance) corresponds to the Price semantic type under context "yahoo_us".

6) **Conversion Functions** – Finally, conversion functions exist to convert values in one context to another. When the Ecoin mediation engine detects that there is a conflict between the modifier values of the source and the user contexts for a particular piece of data being queried, then it applies the conversion functions to automatically convert the value returned from the source to the format and semantics expected by the user who submitted the query.

These elements of an Ecoin Application are defined for each new domain that the Ecoin system is used to perform context mediation for. Defining these elements is the responsibility of the Ecoin Application Engineer. These elements can be defined

23

manually in the native prolog that the core Ecoin system uses, or using tools such as Lee's TextEditor tool to create non-merged applications [2].

2.4.   Merging Ecoin Applications

Firat [4] introduces a process for merging applications for the Ecoin system. However, it must be clarified from the onset that the merging process described by Firat and facilitated by the CLAMP tool introduced in this thesis is *not* a physical merger of Ontologies as has been the goal of much of the research literature.

Before describing the distinction between physical merging and the approach taken by Ecoin, it is instructive to clarify some definitions that will be used when discussing application merging henceforth. Figure 2-7 illustrates the process of application merging, with each oval representing a single Ecoin application used to perform context mediation across a set of data sources. The process of combining two existing applications into a new application (for purposes that we shall discuss later) is referred to as the *application merging process*. In the application merging process, the applications that are merged to form one, new, all encompassing application are referred to as the *child applications* or the *merged applications* (note the "d" at the end of merged). The new, all encompassing application that is created through the merging process is referred to as the *merger application.* When we discuss what parts of an application we will actually be merging, the term "application element" will often be used. *Application Element* simply refers to any of the components that make up an Ecoin application (semantic types, modifiers, contexts, etc.). Our discussion of merging will mainly focus on semantic types, attributes, modifiers, and contexts as the application elements.

A *physical merging* process would describe a process that creates a new application that physically contains two applications, as if those applications were created from scratch in the new application. Instead, the approach that Ecoin takes to merging applications can best be characterized as a *virtual merging* process [3]. With virtual merging, the merger application doesn't physically contain the merged applications. Instead, the merged applications are aligned with regards to their Ontologies and contexts to facilitate a

Figure 2-7: Application Merging Definitions

combined functionality in the new merger application. The key distinction is that virtual merging is strictly used for *functionality* – it is used to include and extend functionality

(with regards to answering and mediating queries) from the child applications that are merged.

The primary goals of virtually merging Ecoin applications are three, as stated by Kaleem [3]:

1) To bring together the sources of the merged applications in to one application. In this way, a user can query sources from both merged applications by submitting queries to Ecoin using the merger application.

2) The second goal of virtual merging is to reconcile the contexts of the merged applications and to union the context mediation capabilities of both merged applications in one merger application. With this objective, which will be referred to as the *cross fertilization of contexts*, mediation of particular types of data that was once available in only one of the child applications is now shared among both applications.

3) The final goal is to extend the newly created merger application where appropriate to include new sources and new mediation capabilities that weren't available in either of the existing merged applications.

We next describe the airfare and car rental applications introduced in chapter 1. These applications and their merger are documented in detail by Kaleem [4]. However, since we will be using Kaleem's example as a vehicle for describing the goals and functionality of the CLAMP tool, we provide an overview of these applications here and give a brief summary of the steps in the merging process of the two applications.

2.4.1. The Airfare Application

2.4.1.1. Sources and Relations

As introduced in chapter 1, the airfare application focuses on websites that deliver information about airline ticket prices. Some sample sources that provide such information include Orbitz, TravelSelect, and ITN. The table below represents the basic data returned by querying these websites after they've been wrapped by the Chameleon web wrapping tool [3]. This data shows prices for tickets from Tokyo (NRT) to Boston (BOS).

In addition to the web sources that provide information about ticket prices, the airfare application uses the Olsen currency conversion website (http://www.oanda.com/convert) to assist in the conversion of currency values. Specialized tables for defining the service fees, paper ticket fees, and visa fees are also defined.

2.4.1.2.   Ontology

The Ontology created to describe the domain of airline ticket reservation is given below. It includes semantic types for all of the data types that exist, and modifiers that capture the context differences that will exist.

Figure 2-8: Sample results from querying airfare data sources (from Kaleem [3]).

Provider	Airline	Destination	Departure	Price
Orbitz	American Airlines 4762	NRT	BOS	977
TravelSelect	American Airlines 4762	NRT	BOS	907.73132
Yahoo	American Airlines	NRT	BOS	966.95
itn	AmericanAirlines	NRT	BOS	4602.7
Expedia	American	NRT	BOS	1045
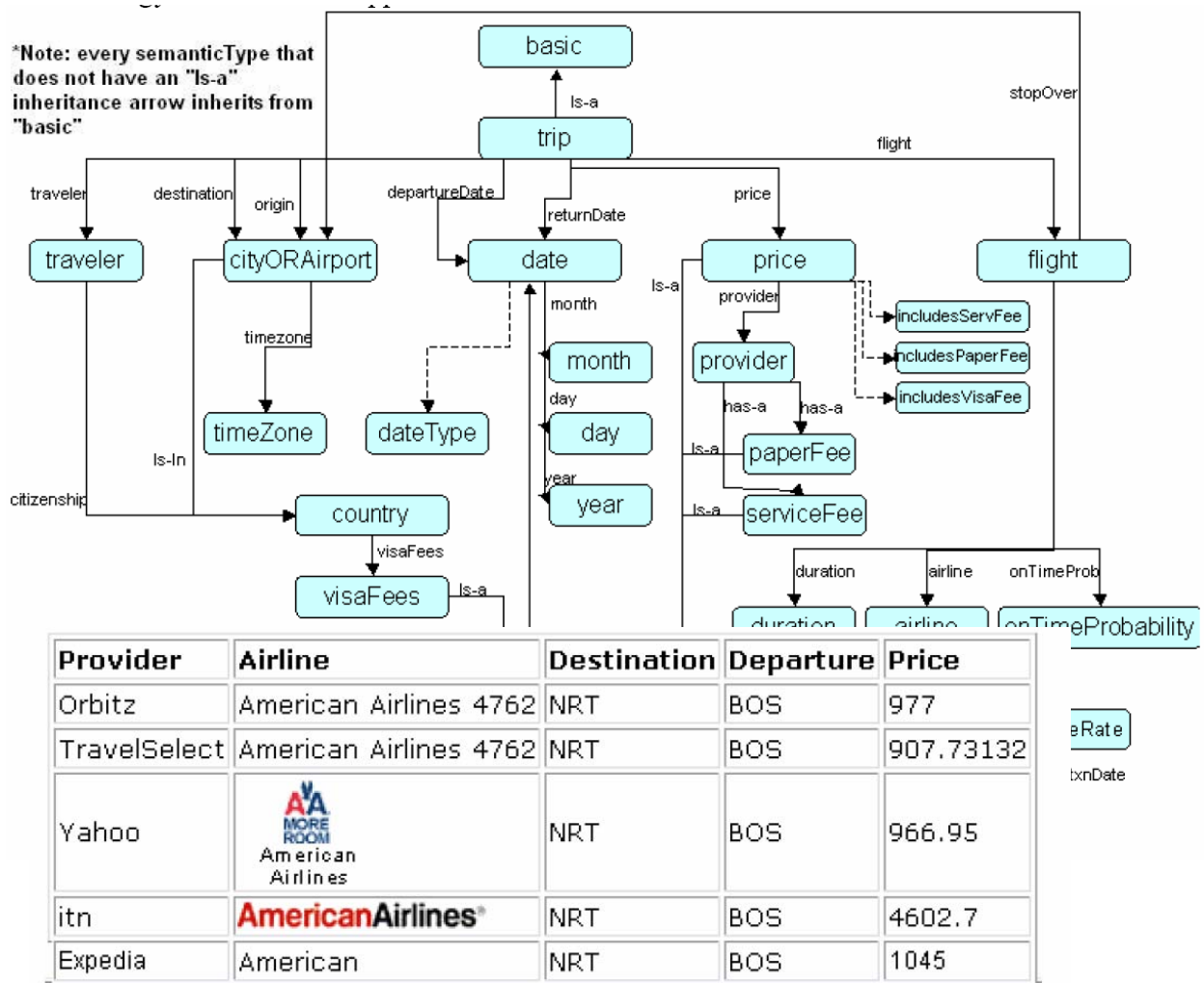
Figure 2-9: Airfare application ontology (from Kaleem [3])

2.4.1.3.   Context

Next, the application needs to have contexts defined for the various sources accessed as well as the users that will be interpreting the data. The table below provides the names and definitions (according to modifier values in the ontology) for the airfare application.

Context Type	Context Name	includesServiceFee	includesPaperTktCharge	includesVisaFee	Currency
Receiver Contexts	Dora's Friend	No	No	No	GBP
	Dora	Yes	Yes	Yes	USD
Source Contexts	Yahoo	Yes	No	No	USD
	Expedia	Yes	No	No	USD
	Orbitz	No	No	No	USD
	Travelselect	No	No	No	GBP
	Itn	No	No	No	USD

Figure 2-10: Airfare application contexts (from Kaleem [3]).

2.4.1.4.   Elevations

The elevations that describe the mappings from columns in the data sources to semantic types in the Ecoin ontology are described in figure 2-11.

Source	Elevations (Column in data source → semantic type)	
Elevations from Airfare sources	- Airline	→ airline
	- Price	→ price
	- Destination	→ cityOrAirport
	- Departure	→ cityOrAirport
	- Date1 (departure date)	→ date
	- Date2 (return date)	→ date
	- Month1 (depart month -some sites use month & day rather than date)	→ month
	- Month1 (return month-some sites use month & day rather than date)	→ month
	- Day1 (depart day-some sites use month & day rather than date)	→ day
	- Day2 (return day-some sites use month & day rather than date)	→ day
	- Provider (i.e. site that found fare, ex. Orbitz)	→ provider
	- IsIn (i.e. the country the destination is in)	→ country
Elevations from currency conversion source	- Exchanged (i.e. fromCurrency) → currency	
	- Expressed (i.e. toCurrency) → currency	
	- Rate → exchangeRate	
	- Date (i.e. txnDate) → date	
Elevations from service fees table	- Provider (i.e. site that found fare, ex. Orbitz) → provider	
	- Service Fee → serviceFee	
Elevations from paper_fees table	- Provider (i.e. site that found fare, ex. Orbitz) → provider	
	- Paper Fee → paperFee	
Elevations from visa_fees table	- Citizenship (of traveler) → country	
	- Destination (of traveler) → country	
	- Visa Fee → visaFee	

Figure 2-11: Elevations from data sources to the Ecoin ontology (from Kaleem [3]).

Conversion Functions

Finally, the Airfare application has several conversion functions defined to deal with the contextual differences described in figure 2-6. Among these conversion functions are:

1) Currency conversion that converts monetary values in one currency to another.

2) Conversion of airline prices that don't include paper fees, visa fees, or service fees to prices that do include these added charges.

2.4.2. The Car Rental Application

2.4.2.1. Sources and Relations

The sources in the car rental application are again several websites – in this case sites that provide information on the availability and pricing of rental cars to user submitted queries. Among the websites used are Expedia Car Rental service (http://www.expedia.com), Yahoo Car Rental service (http://travel.yahoo.com) and QixoCar (http://www.usahotelguide.com/nex-res/cars/).

2.4.2.2.　　Ontology

　　　　The ontology for this application, as defined by Kaleem [3], is shown in the figure below.
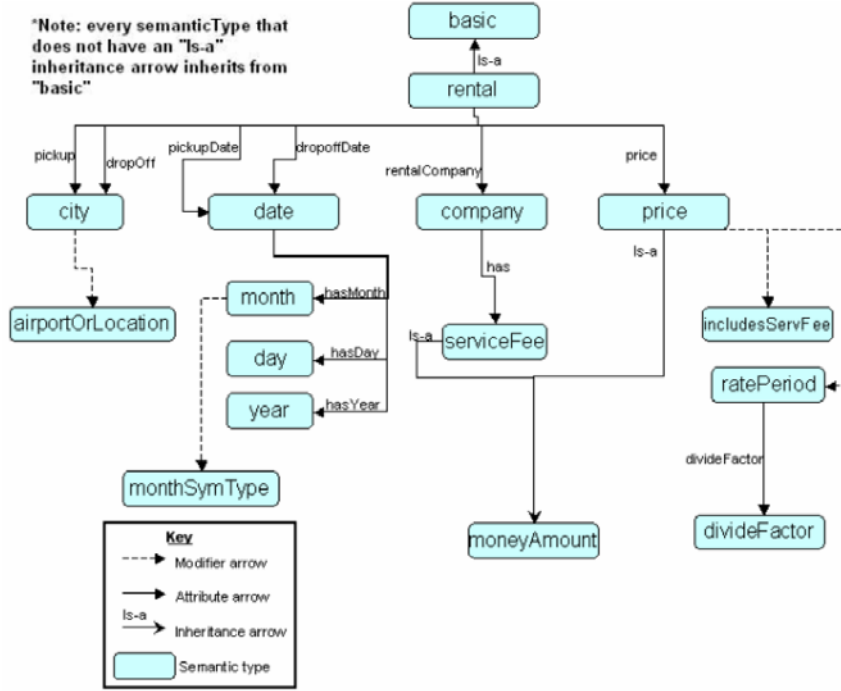
Figure 2-12: The Ecoin ontology for the car rental application.

2.4.2.3.　　Contexts

　　　　The modifier value assignments that define the contexts for the car rental application are described in the table below [3].

Contexts	IncludesServFee	RatePeriod	monthSymType	CityOrAirportCode
Yahoo	"Yes"	dynamic – based on result returned	"three-letter"	"airport"
Expedia	"Yes"	dynamic – based on result returned	"two-digit"	"airport"
Qixo	"No"	dynamic – based on result returned	"two-digit"	"airport"
Joe	"Yes"	dynamic – based on result returned	"three-letter"	"city"
joe's friend	"Yes"	"total"	"two-digit"	"airport"

Figure 2-13: The modifier values defining the contexts for the car rental application.

2.4.2.4.   Elevations

Figure 2-14 below describes the elevations from the sources in the car rental application to the semantic types in the ontology for the car rental application.

Source	Elevations	
Elevations from Airfare sources	- Pickup	→ city
	- Dropoff	→ city
	- Date1 (departure date)	→ date
	- Date2 (return date)	→ date
	- Month1 (depart month -some sites use month & day rather than date)	→ month
	- Month1 (return month-some sites use month & day rather than date)	→ month
	- Day1 (depart day-some sites use month & day rather than date)	→ day
	- Day2 (return day-some sites use month & day rather than date)	→ day
	- Price (rental rate)	→ price
	- Company (i.e. the car rental company)	→ company
	- RatePeriod (i.e. of rate quoted by company)	→ country
Elevations from airport_code_lookup	- Location	→ city
	- AirportCode	→ city
Elevations from month_sym_converter	- Mm_number	→ month
	- symbol	→ month
Elevations from rateperiod_dividefactors	- Rateperiod (ex: 'weekly', 'monthly', etc)	→ ratePeriod
	- DivideFactor	→ divideFactor

Figure 2-14: Car rental application elevations.

2.4.2.5.   Conversion Functions

Among the conversion functions that exist for the car rental application is one to convert from the different codes used to represent localities (whether in city format or airport format) as well as a conversion function to convert the different ways that dates may be represented in the car rental application.

2.4.3.   The Virtual Merging Process Described

With a basic understanding of what makes up an Ecoin application as well as a description of our two applications that will be merged (as described by Kaleem [3] and Firat [4]), we can now discuss what the goals and steps are in the virtual merging process.

To stay consistent with the approach that the CLAMP tool will follow in merging applications, we frame our discussion on virtual merging around the three basic goals we wish to achieve from creating a merger application.

2.4.3.1.   Access to Sources from the Merged Applications.

The first goal of merging two Ecoin applications is to support access to data sources from both Ecoin applications. In merging the airfare and car rental applications, for example, the Ecoin application engineer may want to create a new application where users can query for airline ticket prices and car rental availability at the same time. Without merging the two applications, a user would have to write two separate queries and submit them to two different Ecoin applications to get the data he desires. Such a

goal is desirable with applications for which it will be desirable to access both sets of sources to provide combined information (such as aiding in trip planning by providing access to information about all facets of the trip including airline tickets and car rental information).

2.4.3.2.   Cross-Fertilization of Contexts

The next goal of Ecoin applications is the cross-fertilization of contexts [3]. Through this objective of merging, Ecoin application engineers can use the context capabilities of one application in the other merged application. For example, our airfare application has the capability to reason about currency distinctions across different contexts. The car rental application, however, does not possess this capability.

If we merge the two applications, we may create the possibility of using the currency conversion in airfare to convert currencies in the car rental application. Figure 2-15 demonstrates how creating an isomodifierType relationship the merger application between two semantic types from the child applications does this. The *isomodifierType* relationship denotes an equivalence relation between the semantic types from the two child applications. This equivalence relation essentially means that the two semantic types have the same set of modifiers – explicitly defined in the Ontologies or implicitly extrapolated from observing the roles of the semantic types. By defining this
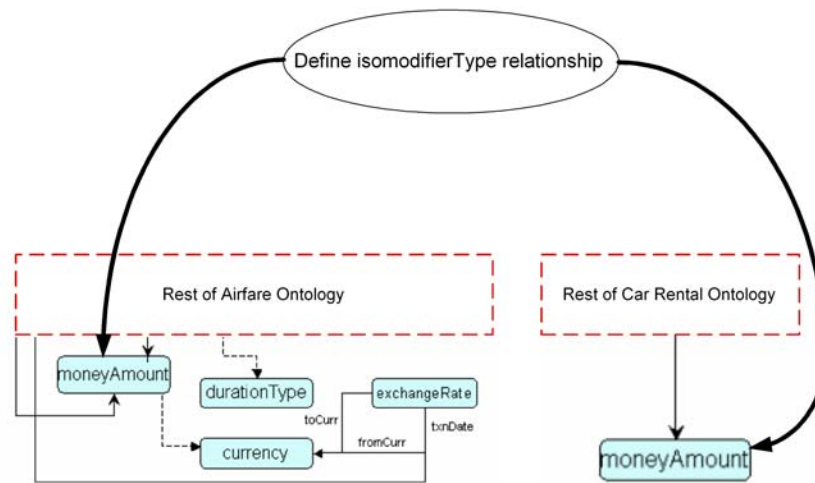
Figure 2-15: Defining isomodifierType relationships between two semantic types in the child applicaitons.

isomodifierType equivalence relationship, the Ecoin application engineer essentially states that the modifiers of the two semantic types be merged in the newly created application.

Once the semantic types from the child applications have been declared as equivalent in the merger application, context mediation capabilities from the two child applications can be shared. The conversion function for currency conversion can now be

used to convert money values originating from car rental data sources, as long as values for the currency modifier have been defined in the merger application for the car rental contexts.

This point about defining modifier values for semantic types that merge the contexts of two child applications brings up an important point. As a result of defining an equivalence relation between the moneyAmount semantic types in figure 2-15, the *implicit* modifier in the car rental application for currency is now an *explicit* modifier in the new merger application. Thus, modifier values must be explicitly spelled out for this modifier that was implicit in the car rental application prior to the merger. This requires the user to cycle through the contexts in the car rental application and assign values for the currency modifier for each context. The discussion on the CLAMP tool in chapter 4 will provide an illustration of how such implicit modifiers are defined.

In addition to declaring semantic types as equivalent, the Ecoin application engineer can also assign equivalence relations between modifier, attributes, and contexts as well. Below is a listing of all four equivalence relations that can be defined when merging two Ecoin applications.

1) **Isomodifertypes** – the equivalence between semantic types discussed above, where the merger application now combines the two semantic types from the merged applications in to one semantic type with a modifier set that is the union of the modifier sets of the individual child semantic types.

2) **Isomodifiers** – this relationship defines two modifiers in the merger application to be equivalent, with the pre-requisite that their domain semantic types have also been declared as equivalent. Isomodifiers combine modifiers that (implicitly or explicitly) share the same modifier value domains. Isomodifier relationships are distinct from isomodifiertype relationships because the former is an equivalence between *modifiers* while the latter is an equivalence relationship between semantic types. In figure 2-15, one could define an isomodifiertype relationship between the two moneyAmount semantic types. If both the moneyAmount semantic types had modifiers specifying currency, they could then be declared as isomodifiers.

3) **Isoattributes** – this is an equivalence relationship stating that two attributes from the child applications are equivalent. In order for such an equivalence to make sense, it would require as a pre-requisite that the semantic types in the domain and range of the attributes be declared as equivalent.

4) **Isocontexts** – Two contexts can be declared as equivalent if they share the same modifier value assignments.

With this discussion on declaring the equivalence relationships between semantic types, modifiers, attributes and contexts, it is a straightforward task to highlight the distinction between virtual merging and physical merging. Virtual merging required such

equivalence relationships to exist because the immutable child applications are contained in the new merger application. With physical merging of the application Ontologies, no equivalence relationships would be required because it would essentially be a process of creating a new ontology (that starts out with all the structures in the child Ontologies), rather than linking the existing Ontologies and contexts through equivalence relationships.

2.4.3.3. Appropriate Extensions to the New Merger Application

The third and final goal/step in creating the merger application is making any new definitions. The user can extend the merger application containing the child application with new sources, new contexts, and new ontology elements where appropriate.

Since we are dealing with a virtual merging process whose semantics involve extrapolating up Ontologies from existing applications rather than creating a new physically merged ontology, there are limits to the operations we can perform to modify the new merger application. For example, semantic types and modifiers from the child applications can not be deleted in the merger application since they are inherited up from applications that already exist in their final form.

This sums up our discussion on some of the background issues related to Ecoin and merging that are crucial to understanding later chapters about the CLAMP tool and guidelines for building Ecoin Ontologies. This chapter introduced the concepts of Ecoin applications and the merging of Ecoin applications and tried to package these concepts in a structure that will make the following discussions easier to present and understand. In addition to the information on Ecoin provided here, details on the Ecoin system can be found in Firat [4] and Lee [2]. Details on merging, and a complete manual walkthrough of the merging of the airfare and car rental applications, can be found in Kaleem [3].

Chapter 3. Design and Implementation of the Context Linking and Merging Process (CLAMP) Tool

3.1. Introduction

The previous chapter introduced the Ecoin application merging process, which is meant to facilitate context mediation of queries that involve data from two existing applications. In order to create a merged application, the Ecoin Application Engineer must follow several stages of development [3]:

1) <u>Making Explicit the Merging of Applications</u> – In this stage, the Ecoin application engineer points out the applications that are being merged. This stage will allow *seamless source access* to the data sources from both applications.
2) <u>Defining Virtual Ontological Structures</u> – Merged applications contain definitions for virtual semantic types, virtual contexts, virtual modifiers, and virtual attributes. These merging-specific structures are not new elements of the ontology and context for the merged application, but rather define mappings between existing ontology elements and contexts from the child applications. These mappings allow *cross-fertilization of contexts*, where context mediation capabilities in one application can now be used to mediate queries to data from the other child application.
3) <u>New Definitions for the Merger Application</u> – Finally, the Ecoin application Engineer may want to define new ontological elements and new contexts for his merger application. This may be to create new context mediation capabilities (such as new context definitions) as well as to handle new sources of data that the merger application will deal with.

The previous chapter demonstrated how this is done manually by defining merger axioms in prolog. Each stage in the development process corresponds to a different set of merger axioms that must be specified to achieve the desired functionality.

Most target users of Ecoin are not experts with prolog, nor should they be expected to understand the merging process in such detail that they can go through and write out the merger axioms for each different stage of the Ecoin application merging process. This is where the motivation for the CLAMP tool comes in. The CLAMP tool provides an intuitive step-by-step interface to the Ecoin application merging process that then automatically generates any necessary merger axioms. The job of the Ecoin application is simplified from writing out merger applications to going through a step-by-step process of defining the necessary relationships that need to exist to merge two applications. Figure 3-1 illustrates the role of the Ecoin application engineer in merging Ecoin applications with and without the CLAMP tool. In the configuration shown in
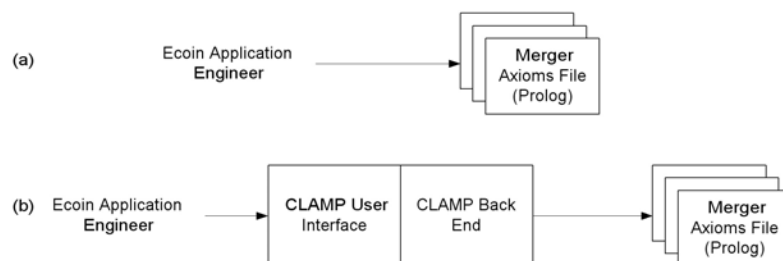

Figure 3-1: Merging Ecoin Application without CLAMP (a) and with CLAMP (b).

34

figure 3-1(a), the application engineer must create the prolog axioms manually, as Kaleem [3] had to do in order to merge applications. However, with CLAMP, the user is guided through a more intuitive user interface, with the actual prolog axioms generated for the application engineer.

Although CLAMP simplifies the merging process to a large degree, it must be stressed that CLAMP isn't meant to be an automatic merging tool. Instead, CLAMP is designed to ease the process of merging, which still is ultimately guided and defined by a human user. Furthermore, providing guidance on the merging process is only the most basic goal of CLAMP. There are several other objectives of merging applications through CLAMP that are important in the design and implementation of CLAMP:

1) Support merging as a *hierarchical process*, where merger applications themselves can be merged with other applications. Figure 3-2 illustrates this goal by showing two applications, App1 and App2, that are merged to create a third application, Mapp2. Mapp2 is then merged with App3 to create yet another merged application, Mapp1.
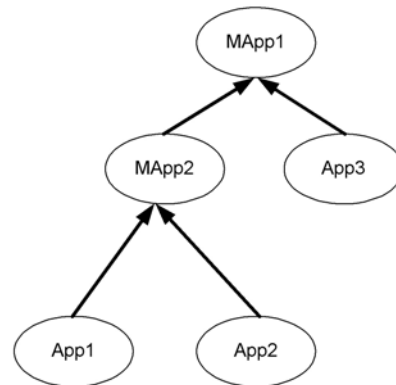
Figure 3-2: Hierarchical merging.

2) As Firat [4] points out, the merging process with Ecoin applications is meant to be a *two at a time process*. This means that only two applications can be merged in to a single merger application. In order to merge more than two applications, hierarchical merging can be used.

3) Merging applications should not require modification to the existing child applications. When an application that merges two existing applications is created, the only dependencies that are introduced must be from the merger application to its children. The existing child applications should not have any dependency on the merger application.

4) Ideally, the merging process should become easy and portable. Different user communities should be able to share their Ecoin applications. One should be able to take existing applications that have been created by one set of users and be able to merge it or extend it to support his own needs. CLAMP should facilitate this process rather than hindering it.

5) The CLAMP tool should be integrated with the existing tools for creating Ecoin applications. This is an important priority because part of the merging process involves creating new Ecoin application structures, which is done through the existing infrastructure. Thus, the CLAMP tool must be integrated with this infrastructure, which is the system for metadata management and representation described by Lee [2].

3.2. Unique Naming for Application Merging

As mentioned in the previous chapter, one of the most important prerequisites that make merging possible is a unique naming scheme for Ecoin applications and the different elements that make up an Ecoin application. In particular, the following should have unique names within the Ecoin system in order to avoid ambiguity when merging:

- **Ecoin applications** – it must be possible to uniquely identify the child applications as well as the merger application.

- **Ontology Elements** – this includes semantic types, modifiers, and attributes.

- **Contexts**

Each module in the Ecoin system – from user interface, to Coin Model data structures, to persistent file representations of Ecoin applications – needs to implement this unique naming scheme. The specific implementation of naming for each of these modules shall be discussed in the relevant sections later in this chapter.

The goal of this section is to describe how the naming scheme applies abstractly to an Ecoin application. Naming has been implemented in such a way that any string based scheme for assigning Universal Resource Identifiers (URIs) can be applied to the Ecoin applications. Each Ecoin application itself is assigned a domain URI as well as an application name. Thus, an application can be identified uniquely using both the name and the URI. As long as the domains of applications are globally unique, and the names of applications are unique within a given domain (i.e. installation of the Ecoin system), then there is a guarantee that there will be no ambiguity in naming of Ecoin applications.

Figure 3-3 illustrates this concept by showing several different installations, each with their own globally unique URI, as well as individual applications within those installations that are unique within the scope of each installation. Even though there is a "CarRental" application existing in both domains, they can be distinguished because of their unique domain URIs. If, for some reason, the Singapore Car Rental application had
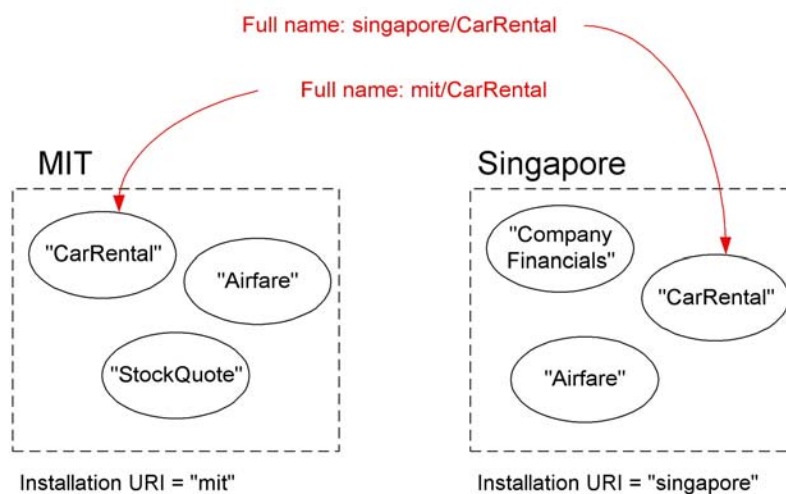
Figure 3-3: An example of naming Ecoin Applications across multiple installations.

to be moved to the MIT domain in order to merge it with the existing Car Rental application, there would be no fear of naming conflicts because the applications are defined by their URI as well as their names.

These restrictions on naming are ideal since it is reasonable to expect users from one Ecoin installation (say at MIT) to assign unique names to their applications, but that may not necessarily be the case with applications developed separately in many different installations. This thesis does not attempt to address the problem of globally unique naming, although such schemes are definitely feasible (as evidenced by existing arrangements such as Universal Resource Locators for web pages across the world).

For ontological elements contained within individual Ecoin applications, the name consists of the complete name of the containing Ecoin application, as well as the name for the element itself. Thus, the containing application name serves as a URI for the element, while the name identifies the element uniquely within the scope of the application. With ontological elements, as long as the complete names of the Ecoin applications in an Ecoin installation are unique and the names of ontological elements *within* any given application are unique, then there is no risk of ambiguity of names when merging two applications. So, if two applications are being merged and they both have a semantic type named "price", there will be no ambiguity because the semantic type has a different URI pre-pended to it depending on the containing application.

3.3. Design Overview of CLAMP

We now turn to looking at how CLAMP fits in to the existing Ecoin system architecture, as well as introducing the main modules of the tool that I shall discuss in detail in subsequent sections.

Figure 3-4 gives an overview of the modules that make up the CLAMP tool. Each 3-dimensional structure in figure 3-4 represents a module within the Ecoin metadata

Figure 3-4: Overview of the CLAMP/Application Editor Architecture.

37

generation and management system. The prolog axioms, Ecoin registry, and Ecoin application elements represent files that are created or used by the system. As the figure points out, the arrows are meant to describe dependencies between different modules and files.
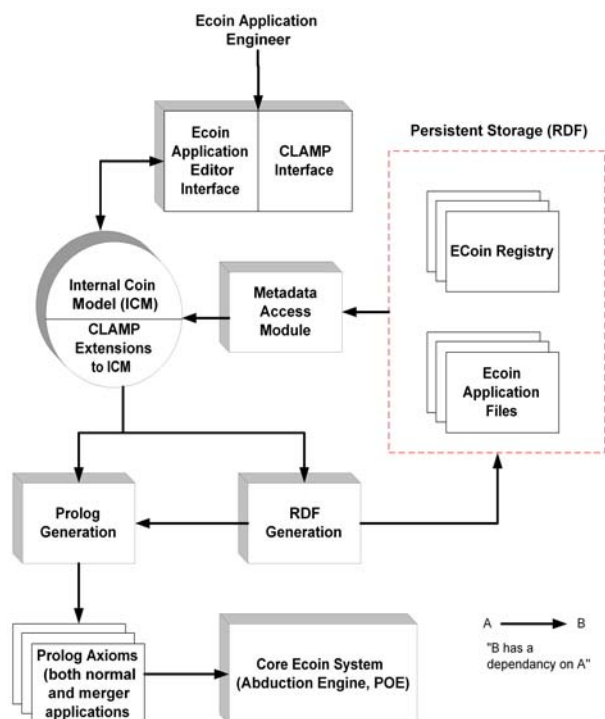
The design for CLAMP essentially involves an extension to the existing system created by Lee [2] for Ecoin metadata management. This was done instead of creating a completely independent tool because one of the key objectives of CLAMP is to extend merger applications with new ontology, context and source information. In order to provide integration with the existing Ecoin infrastructure and the ability to take advantage of existing tools for creating new (non-merged) Ecoin applications, it was necessary to couple the design of CLAMP with the Application Editor designed by Lee [2]. Although this complicates the implementation process and increases dependency of CLAMP on existing tools, I believe it is worth the sacrifice in order to simplify one of the main phases of the merging process – namely creating new ontology and context definitions for the merger application.

3.3.1. Definitions

It is important to make explicit some definitions that will be used throughout the remainder of this chapter. The term *original application editor* or *application editor* refers to the metadata management system originally designed and implemented by Lee [2]. It includes the user interface, the Internal Coin Model, the RDF and Prolog generation modules and the module for loading the ICM from the metadata representations. The *CLAMP extensions to the application editor* or *CLAMP tool* will refer to the extensions of the application editor made to support merging Ecoin applications. *Ecoin metadata* refers to the RDF representation of the Ecoin application stored in file format.

3.3.2. Ecoin, the Application Editor and the Clamp Editor: Big Picture

Although figure 3-4 provides a design overview of CLAMP, it may still not be clear exactly how the Ecoin system functions with the application editor or the new CLAMP tool that extends the application editor. To make this point concrete, refer to figure 3-5, which again shows the Ecoin system in full as introduced in chapter 2.

The application editor and the CLAMP tool have been created to aid the Ecoin application engineer in creating new Ecoin applications. These applications are defined by Prolog axioms in files used by the core Ecoin system. These axioms describe the Coin Model circled in figure 3-5. Thus, the final objective of the application editor and the CLAMP tool are to generate the Prolog axioms that specify the Coin Model for an application. Once the Ecoin application engineer has specified the Coin Model and the data sources, Ecoin users may submit queries that are context mediated by the Ecoin system.
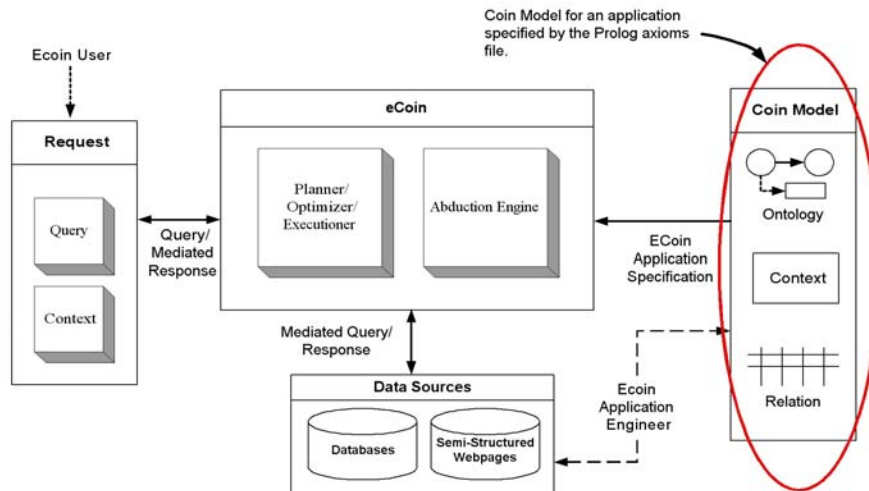
Figure 3-5: The prolog axioms generated by the Application Editor and the CLAMP tool (for merging applications) comprise the Coin Model used by the core Ecoin system.

3.3.3. The Data Necessary to Represent an Ecoin Application

Many of the different files and modules within the system exist to perform essentially the same function: to provide a representation of the Ecoin application being created. These representations include the Prolog Axioms file, the Ecoin application Files, and the Internal Coin Model. Several representations exist because they each have a different role in defining an Ecoin application within the overall system. These roles will be described in the next sections of this chapter, where we discuss each in detail.

It is instructive, however, to specify the basic information that each representation is trying to define. The information represented by Lee's Application editor is:

1) The name and URI of the application itself.

2) The **semantic types** of an application, including information about:
    a. URI and name of the semantic type
    b. Inheritance information.
    c. The set of attributes that exist for that semantic type.
    d. The set of modifiers that exist for that semantic type.

3) The **attributes** of an application, including information about:
    a. URI and name of the attribute.
    b. The domain and range of the modifier (both of which are from the set of semantic types in the application).

4) The **modifiers** of an application, including information about:
    a. URI and name of the attribute.
    b. The domain and range of the modifier.

39

      c. A table mapping contexts to particular modifier values. A context is thus defined by the set of values assigned to the modifiers in the application.

      d. The definition of the conversion function for each modifier that specifies how data is converted between contexts when that modifier has differing values for two contexts.

5) The **contexts** of an application, including information about:
      a. URI and name of the context.
      b. Any contextual inheritance relationships.
      c. The set of elevations that map a column from source relations (see below) with a context and a semantic type in the application ontology.

6) Information about the **sources**, **source relations** (mappings from a provided source to a representation as a relational table that Ecoin can use), **elevations** mapping source columns, contexts, and semantic types is also encoded within the different modules of the system. I will leave a detailed discussion of these elements to Lee [2] since the merging process is not dependent on a firm understanding of their exact representation.

This representation must be extended in order to allow for merging of Ecoin applications. The following are the data structures that must be defined for CLAMP in order to successfully facilitate merging:

1) The **merged** or **child** applications that are being combined to make the new merger application. Since merging in Ecoin is currently a two at a time process, this will be a set of two applications.

2) **Virtual semantic types** that store information about semantic types that are declared as "equivalent".

3) **Virtual attributes** that store information about attributes from child applications that are declared as "equivalent".

4) **Virtual modifiers** that encapsulate modifiers from the child applications that are declared as "equivalent".

5) **Virtual contexts** that contain contexts from child applications that are declared as "equivalent".

Each representation described in the next sections will be organized around this basic set of information that must be provided in order to successfully represent an Ecoin application and to describe the merging relationships that exist when linking two Ecoin applications.

3.3.4. Description of Overall Design Interdependencies

Before giving a detailed description of what the key CLAMP editor modules do, I will first give an overview of how the different modules interact together to perform CLAMP's basic function – helping Ecoin application engineers build merged applications. At the heart of the system is the Internal Coin Model (ICM). The ICM is an intermediate representation of the Ecoin application. The user interface depends on the ICM to display the existing application information to the user (the state of the ontology, the existing contexts, etc.). When an application is created from scratch, the ICM is populated as the user defines new elements of the application. When an application is loaded from persistent storage, then the ICM is populated by parsing the application's specification from RDF.

This specification is contained within a set of files that have been collectively named the "Ecoin Application Files" in figure 4-4. These files specify an RDF graph that also represents the relevant application information introduced above. The Ecoin Registry is a set of files that specifies which applications exist at a particular installation of Ecoin and where the RDF application files reside.

The other set of files generated by CLAMP are the Prolog axioms files that describe the Ecoin application in Prolog. The core Ecoin system uses the Prolog axioms files to mediate queries. Each application has one Prolog file that defines all of the ontology, context and source information described above.

The generation of both the RDF application files as well as the Prolog axioms files rely on the representation of the merged application given by the ICM. The RDF generation module iterates through all of the ontology, context and source information in the ICM to create RDF metadata files. The Prolog generation module translates some application information directly from the RDF metadata files, and also retrieves application information from the ICM.

It should now be clear why the CLAMP tool and application editor tools exist, where they fit in with the overall Ecoin system, what the organization is of the modules that compose both tools, and the interdependencies each of these modules have with one another. The application editor is described in detail by Lee [2]. The next sections describe the design and implementation of the extensions to Lee's tool to support merging. Note that a detailed description of the user interface of CLAMP is given in the next chapter, where I discuss how an Ecoin application engineer would navigate the tool to create merged applications.

3.4. The Internal Coin Model for Merged Applications

The Internal Coin Model is at the center of the design of the application editor and the CLAMP tool. The user interface, RDF metadata, and the Prolog axioms all depend upon the data structures and algorithms implemented within the ICM. I first describe the data structure of the ICM, with a detailed focus on the extensions implemented for

CLAMP. Then, I describe how applications and application elements are named within the ICM to satisfy the unique naming requirement needed for merging. The last critical extension to the ICM that requires discussion is the implementation of algorithms to provide a *logical view* of the semantic types, attributes, modifiers, and contexts that exist within a merged application. These algorithms are based on Firat's definition of what the logical view of an application is [4].

3.4.1. The Internal Coin Model Data Structure

Figure 3-6 outlines the data structure of the ICM. The extensions for CLAMP are highlighted.

Coin_Model
id:string
merges:array(Coin_Model)
name:string
uri:string
SemanticTypes:array(Ont_SemanticType)
Contexts:array(Cxt_Context)
Relations:array(Src_Relation)
virtualSemanticTypes:array(Ont_Virtual-SemanticType)
virtualContexts:array(Ont_VirtualContext)
isMerged:boolean

Ont_SemanticType : NamedApplicationElement
parent:Ont_SemanticType
Attributes:array(Ont_Attribute)
Modifiers:array(Ont_Modifier)

Ont_Attribute : NamedApplicationElement
name:String
from:Ont_SemanticType
to:Ont_SemanticType

Ont_Modifier : NamedApplicationElement
name:String
from:Ont_SemanticType
to:Ont_SemanticType
ModifierContextValue:hashtable(Cxt_Context, array(Ont_Attribute|string))
ConversionFuntion:string

Cxt_Context : NamedApplicationElement
name:String
parent:Cxt_Context
modifiers:array(Ont_Modifier)
elevatedRelations:array(Src_ElevatedRelation)

NamedApplicationElement
name:string
uriPrefix:string

Ont_VirtualSemanticType : Ont_SemanticType
isomodifierTypes:array(Ont_SemanticType, Coin_Model)

Ont_VirtualAttribute : Ont_Attribute
isomodifiers:array(Ont_Modifier, Coin_Model)

Ont_VirtualModifier : Ont_Modifier
isomodifiers:array(Ont_Modifier, Coin_Model)

Cxt_VirtualContext : Cxt_Context
isocontexts:array(Cxt_Context, Coin_Model)

Src_Relation

Src_ElevatedRelation

Src_Column

Figure 3-6: The ICM data structure.

Lee [2] provides a detailed discussion of the original application editor data structures. The Src_Relation, Src_ElevatedRelation, and Src_Column classes are not expanded as they are not changed for merging Ecoin applications. The extensions to incorporate all of the information necessary to describe a merged application (as discussed in section 3.3) are:

1) The merged or child applications of a merger application are stored in the Coin_Model.merges array. This is an array of Coin_Model objects, which represents the Ecoin application.

2) Ont_VirtualSemanticType represents the virtual semantic types in the merger application. This class inherits from Ont_SemanticType so that virtual semantic types can be treated as normal semantic types when necessary in some contexts (such as displaying the complete set of semantic types that exist for an application). Ont_Virtual SemanticType has a data field called isomodifierTypes. This is an array that stores the child semantic types that have been declared as equivalent within the merger application.

3) Ont_VirtualAttribute represents the virtual attributes in the merger application. It inherits from Ont_Attribute so that virtual attributes and normal attributes can be referenced in the same context for reasons similar to those described for Ont_VirtualSemanticType.

4) Ont_VirtualModifier represents the virtual modifiers in the merger application. It inherits from Ont_Modifier in order to allow normal and virtual modifiers to be referenced in the same context.

5) Cxt_Context represents the virtual contexts in the merger application and inherits from Cxt_Context so that normal and virtual contexts can be references together.

The set of virtual semantic types and virtual contexts in a merger application are stored in the Coin_Model.virtualSemanticTypes and Coin_Model.virtualContexts arrays respectively. Since Ont_VirtualSemanticType inherits from the normal semantic type class, it stores the set of modifiers (virtual and non-virtual) that are defined for a virtual semantic type. Similarly, Ont_VirtualSemanticType also has an array to store all of the attributes (virtual and non-virtual) defined for each virtual semantic type.

3.4.2.   Unique Naming in the ICM

Unique naming at the ICM level requires a mechanism to attach URIs to applications and application elements. The Coin_Model class represents the whole Ecoin application. Thus, a "uri" field is added to this class and the complete name of the application at the ICM level consists of the URI and the name of the Coin_Model object.
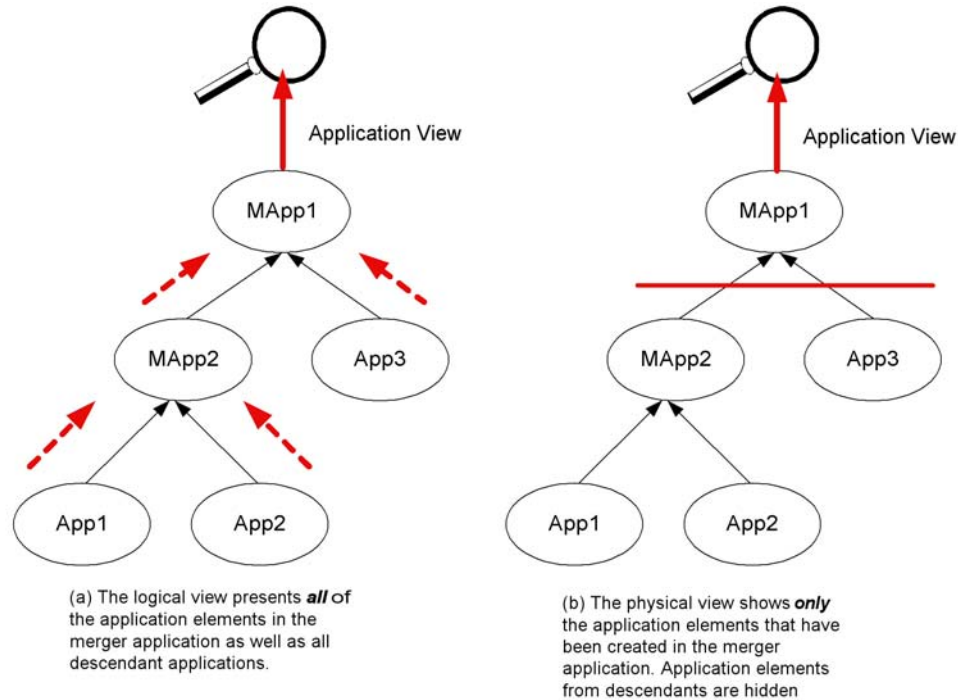
For application elements, a class called "NamedApplicationElement" has been created that contains the name and URI of an application element. All the application element classes (Ont_SemanticType, Ont_VirtualSemanticType, Ont_Modifier, etc.) inherit from this class and thus contain a name/URI pair.

3.4.3. Physical and Logical Views of the Application at the ICM Level

For the purpose of merging applications, the ICM isn't just an intermediate representation of the merger application. It also implements algorithms that provide logical and physical views of application elements. The *physical view* of an application contains all the elements explicitly declared for an application. The *physical view* only requires looking at the state of the application in question and disregards any information about what application elements exist in the children of a merger application.

The *logical view* of an application is equivalent to the physical view for normal applications. However, for merger applications, the logical view includes application elements from both the merger application as well as child applications. Figure 3-7 illustrates the distinction between logical and physical views. In the case of hierarchical merging, as figure 3-7 shows, application elements from all descendant applications may be displayed.

Figure 3-7: Difference between the logical and physical view of an application.

(a) The logical view presents *all* of the application elements in the merger application as well as all descendant applications.

(b) The physical view shows *only* the application elements that have been created in the merger application. Application elements from descendants are hidden

Why is there a need to have a physical and logical view when merging applications? Merging applications involves linking semantic types, modifiers, attributes, and contexts from descendant applications. When two normal applications are merged, this simply involves looking at their physical application elements and linking those. However, when the merging hierarchy goes deeper (that is, we are merging applications that are themselves merger applications), there needs to be a consistent view of all of the application elements available for linking. The physical view is so named because it shows only the application elements that are physically a part of that application. The logical view, on the other hand, provides a view of all of the application elements that can

be inferred as part of the application by descending down the merging hierarchy for the application.

An example can help clarify this point. Suppose that an Ecoin application engineer is creating a merger application, Mapp1, from two existing applications: Mapp2 and App3 (shown in figure 3-7). Mapp2 itself is a merger application with child applications App1 and App2. Because of upward inheritance, all semantic types in App1 and App2 are also a part of Mapp2 (unless they've been declared as isomodifiertypes of one another, in which case only one of the two isomodifiertypes is part of the merger application, Mapp2). The application engineer who is creating Mapp1 should be able to link not only the physical semantic types from Mapp2 and App3, but also the semantic types inherited from App1 and App2 by Mapp2. Providing a consistent view of all the semantic types available in Mapp2 that are available for linking requires implementing the physical and logical views for applications. A similar example applies for modifiers, contexts, and attributes.

As I alluded in the previous paragraph, not all semantic types from descendants are upward inherited and become a part of the logical view of a merger application. If two child semantic types are declared equivalent in the merger application, they are excluded from the logical view and the virtual semantic type containing them is shown instead. Firat [4] describes precisely what elements the logical view of an application contains. He gives definitions for what comprises the logical set of semantic types, contexts, modifiers, and attributes for an application.

3.4.4.   An Algorithm For Presenting the Logical View of an Application

The pseudo-code below shows a generic algorithm that provides the application elements in the logical view of an application as defined by Firat [4]. This is the basic algorithm used by the ICM module to return all semantic types, attributes, modifiers, and contexts in an application. I will first discuss how the algorithm works, and then provide detail on how it is implemented for semantic types, attributes, modifiers, and contexts.

```
//GetPhysicalApplicationElements() returns the set of all application elements
//explicitly defined for this application.
Set GetPhysicalApplicationElements() {
    Return ApplicationElementSet UNION VirtualApplicationElementSet
}

//Returns true if the given application element is mapped to a virtual
//application element in this application
Boolean Mapped(ApplicationElement) {
    Set AllPhysicalApplicationElements = GetPhysicalApplicationElements()

    For Each Virtual Application Element, vAppEl, in
    AllPhysicalApplicationElements {
        if vAppEl links ApplicationElement then Return true
    }
    Return false
}

//GetLogicalApplicationElements() returns the set of all application elements
//in the logical view of this application.
Set GetLogicalApplicationElements() {
    Set returnSet = GetPhysicalApplicationElements()

    For Each Child Application, child, in this Application {

        //recursively call GetLogicalApplicationElements() on child apps
        Set childApplicationElements = child.GetLogicalApplicationElements()

        For Each Child Application Element, childAppEl, in
        childApplicationElements {
            If NOT Mapped(childAppEl) {
                Add childAppEl to the returnSet
            }
        }
    }

    Return returnSet
}
```

The first function, GetPhysicalApplicationElements( ), gets the set of application elements that are explicitly defined for the application. Mapped( ) is a function that returns true if there is an equivalence relationship (linking) defined in the current application involving the application element provided as the argument. GetLogicalApplicationElements( ) then constructs the view of all of the logical application elements for the current application. It starts with the set of physical application elements. It then recursively finds the logical application elements for each of its child applications (if the current application is a merger application). The recursion stops at descendant applications that are non-merged applications, whose logical view is equivalent to their physical view. With this set of logical application elements in hand from the children, the algorithm then proceeds to add each child logical application element that is not already mapped in the current application.

This algorithm is used to fetch all the semantic types and contexts for a given application. For modifiers and attributes, the algorithm is used to return the set of all modifiers and attributes for an individual semantic type within the current application.

The CLAMP user interface uses the GetLogicalApplicationElements algorithms to show the semantic types, attributes, modifiers, and contexts available to the Ecoin application engineer for merging. When describing the merged application in RDF and Prolog, however, only information about what is explicitly defined for the given application is needed. Thus, those modules use the physical view of the ICM representation to represent the application.

3.5. RDF Metadata for Merged Applications

A persistent representation of merged (and non-merged) applications is stored in RDF. This consists of registry files containing information about the identification and structure of the application, as well as application files that define all of the application elements that make up the application.

RDF, which stands for Resource Description Framework, is a language for representing information about resources on the World Wide Web [5]. RDF basically represents a graph consisting of nodes and arcs. It can be specified in a variety of formats, but the most common format used is XML. This is the format that Ecoin uses to describe applications in RDF.

As mentioned earlier, the RDF representation of an application is derived from the ICM representation. Refer to Lee [2] for a complete description of the structure of the Ecoin registry and application files for normal applications. The discussion below will concentrate mainly on the extensions made to the Ecoin registry and application files to support merging.

The Ecoin registry has been extended as follows to include information necessary for merged applications:

- The application is identified by an application id, name *and* URI.

- There is a field for each application in the registry specifying if it is a merged application.

- If the application is a merged application, there are two fields to specify the child applications.

The registry points to the application files describing the different elements that make up an application. The organization of these files has not been changed for merging.

The application files adhere the RDF schema summarized in figure 3-8. This schema is close to the data structure representation described for the ICM. Each set of application files (that completely describe a single application) is an instance of this RDF schema. The CLAMP extensions are highlighted in the figure. The original Ont_SemanticType, Ont_Attribute, Ont_Modifier, and Cxt_Context resource classes have been extended to include URIs for the application elements. LinkedSemanticType, LinkedAttribute, LinkedModifier, and LinkedContext are used as container classes to represent application elements that have been linked in the merger application.

Application
• ApplicationOntology:string
• ApplicationContext:string
• ApplicationSource:string
• ApplicationMisc:string

Ont_SemanticType
• Ont_SemanticTypeName:string
• *Ont_SemanticTypeURI:string*
• Ont_SemanticTypeParent:Ont_SemanticType

Ont_Attribute
• Ont_AttributeName:string
• *Ont_Attribute_URI*
• Ont_AttributeFrom:Ont_SemanticType
• Ont_AttributeTo:Ont_SemanticType
• Ont_AttributeElevationFunction:string

Ont_Modifier
• Ont_ModifierName:string
• *Ont_ModifierURI:string*
• Ont_ModifierFrom:Ont_SemanticType
• Ont_ModifierTo:Ont_SemanticType
• Ont_ModifierConversionFunction:string
• Ont_ModifierContextValues:Ont_ModifierContext-ValuePair

Ont_ModifierContextValuePair
• Ont_ModifierContext:Cxt_Context
• Ont_ModifierStaticValue:string
• Ont_ModifierDynamicValue:Ont_Attribute

Cxt_Context
• Cxt_ContextName:string
• *Cxt_ContextURI:string*
• Cxt_ContextParent:Cxt_Context

Ont_VirtualSemanticType : Ont_SemanticType
• *isoModiferTypes : LinkedSemanticType*

Ont_VirtualAttribute : Ont_Attribute
• *isoAttributes : LinkedAttribute*

Ont_VirtualModifier : Ont_Modifier
• *isoModifiers : LinkedModifier*

Ont_Virtual Context : Ont_Context
• *isoContexts : LinkedContext*

LinkedSemanticType
• *LinkedSemanticTypeName*
• *LinkedSemanticTypeAppID*
• *LinkedSemanticTypeAppURI*
• *LinkedSemanticTypeAppName*

LinkedAttribute
<properties analogous to LinkedSemanticType>

LinkedModifier
<properties analogous to LinkedSemanticType>

LinkedContext
<properties analogous to LinkedSemanticType>

Src_Relation

Src_Column

Src_ElevatedRelation

Src_ElevatedRelationColumnSemanticTypePair

Misc_HelperFunction

Figure 3-8: The RDF schema for merged Ecoin applications.

The registry files and the RDF application files combine to describe all of the necessary information needed for merger applications, as discussed in 3.3.3. Note that Ont_VirtualSemanticType, Ont_VirtualAttribute, Ont_VirtualModifier, and Ont_VirtualContext all inherit from their non-virtual counterparts. Adding name and URI information for these classes is thus not necessary.

1)   The merged or child applications are specified in the registry files for the merger application.

2)   The Ont_VirtualSemanticType class is used to represent virtual semantic types. It will include two isoModifierType properties in RDF that reference the semantic types that are linked.

3)   The Ont_VirtualAttribute class is used to represent virtual attributes, and includes isoAttributes properties to reference the attributes that have been linked.

4)   The Ont_VirtualModifier class represents virtual modifiers and includes the isoModifiers property to reference to the linked modifiers.

5)   The Ont_VirtualContext class represents virtual contexts and includes the isoContexts property to reference to the linked contexts.

3.6.   The Prolog Axioms for Representing Merger Applications

Finally, we turn to the prolog axioms that are the third form of representing a merger application. The Prolog axioms are also generated using the ICM representation of the merged application. Unlike the ICM and RDF metadata representations, the prolog axioms do not create new "virtual" application elements. Rather, the prolog axioms describe the equivalence relationship between the application elements from the child applications that have been linked. The abduction engine then appropriately interprets this equivalence relationship in order to mediate queries. The "virtual" elements that were necessary in the ICM and the RDF metadata representations can be thought of as containers for this equivalence relation.

The following list illustrates how the prolog axioms represent the information needed to describe a merger application:

1) The prolog axiom for referring to the two child applications being merged is:
   ```
   'rule(merges([application1, application2]), (true)).'
   ```

   "1" and "2" at the end of "application" above are replaced by the appropriate application ID (assigned by the Ecoin system for each installation) for the child applications.

2) The following axiom defines an isomodifierType relationship between two semantic types, semType1 and semType2 from application1 and application2 respectively. The name for semType1 is used in the merger application, while that of semType2 is defined as equivalent. Thus, any new ontology structures defined must be done using the semType1 name.

```
'rule(isomodifiertype(application1, application2,
semType1, semType2), (true)).'
```

3) The following axiom defines an isomodifier relationship between two modifiers, mod1 and mod2. mod1 is the name used in the merger application.

```
'rule(isomodifier(application1, application2, mod1,
mod2), (true)).
```

4) The following axiom defines an isoattribute relationship between two attributes, attr1 and attr2. attr1 is the name used in the merger application.

```
'rule(isoattribute(application1, application2,
semType1, semType2), (true)).
```

5) The following axiom defines an isocontext relationship between two contexts, cxt1 and cxt2. cxt2 is the name used in the merger application.

```
'rule(isocontext(application1, application2, cxt1,
cxt2), (true)).
```

Chapter 4. CLAMP Tool User Interface

This chapter will provide CLAMP tool users with a step by step guide on the user interface. The example used henceforth will be the combined airfare and car rental application that was merged manually by Kaleem [3] and introduced earlier in chapter 2.

As we discussed in chapter 3, the role of CLAMP is to provide a consistent user interface that guides the Ecoin application engineer through the process of merging Ecoin applications (2 at a time). Figure 4-1 below re-illustrates this point about CLAMP. As the figure shows, the output of CLAMP is an automatically generated merger axioms
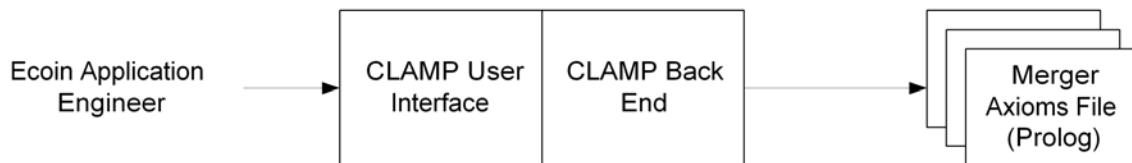

Figure 4-1: The CLAMP tools role in facilitating merging of Ecoin applications.

file that can then be provided to the Ecoin system to perform mediation on the sources from both child applications as well as any new sources the user may have wanted to add. CLAMP is *not* meant to be an automatic merging tool. It includes features (such as the physical and logical views of applications) that make the merging process easier and more intuitive, but user input is still required to specify the relationships necessary to merge the child applications.

CLAMP was designed to work with Microsoft Internet Explorer, as is the case with Lee's Application Editor Tool which provides the same web interface as CLAMP [1]. The process of installing and running CLAMP are as follows:

1) The machine that actually runs the CLAMP software needs to have several properties. The first of these is that it should be running Microsoft Windows XP professional edition with built in Internet Information Services (IIS) software. This software will be used to serve up the dynamic pages that make up the CLAMP user interface.

2) In addition to IIS, the Ecoin application engineer should have Microsoft Visual Studio .Net installed to run the CLAMP/Application Editor software.

3) The CLAMP files include a directory called "AppEditor". This complete directory must be installed in the $IIS-WWWROOT/ directory (the directory where all IIS web pages are contained).

4) Once CLAMP has been installed, you can run it through Microsoft Visual Studio .Net by loading the AppEditor files and selecting "start without debugging" from the Visual Studio menus.

5) Finally, once the software is running, it can be accessed through http://$LOCALHOST-NAME/appEditor/TextInterface.aspx?location=$REGISTRYNAME. The $LOCALHOST-NAME variable represents the domain of the machine that CLAMP is installed on. The $REGISTRYNAME is the name of the Application Editor registry being used to organize all the Ecoin applications on the installation.

4.1. Creating and Loading Merged Applications

Once CLAMP is installed, the first page the user sees is the initial Application Editor menu shown in figure 4-2. Items of note have been circled and numbered and will be discussed below according to their numerical assignments. This page is used to load and create new applications – merged and non-merged. It is an extension of the initial application loading and creating page created by Lee [2] for the Application Editor.
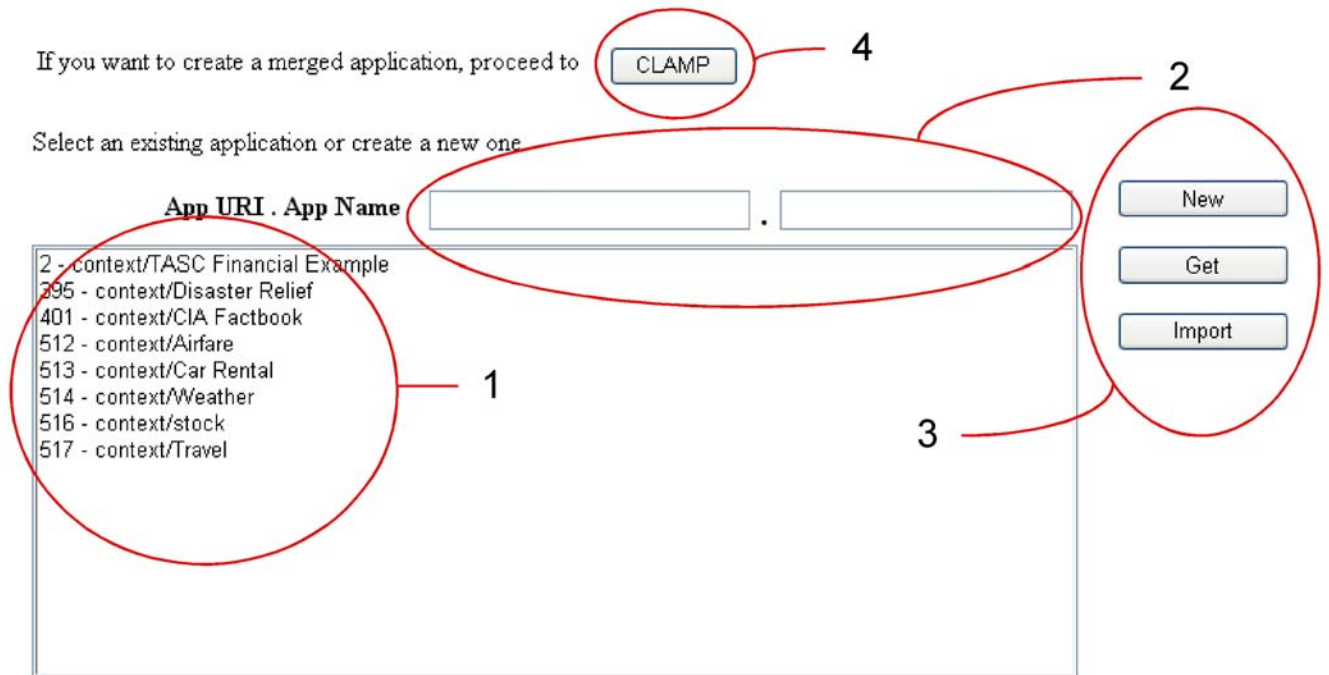

Figure 4-2: TextInterface.aspx, from which you can load existing applications or create new Ecoin applications.

1) This is a listing of the Ecoin applications that are currently installed in application Editor. These applications can be loaded and then edited using the application Editor. This list includes normal Ecoin applications and merger applications (context/Travel is an example of this). The applications are listed according to their application Ids, the installation URI (context in the example above) and the application name.

2) If the user wants to create a new *non-merged* application, he must first specify the installation URI as well as the name for the new application in item2 circled above.

3) The three buttons in item 3 are used to create new *non-merged applications* and load both merged and non-merged applications. In order to create a new non-merged application, the user inputs the URI and name in item 2 and then clicks the "New" button, which takes the user to the normal Application Editor interface created by Lee [2]. If the user selects the "Get" button after highlighting one of the applications in item 1, then one of two things can happen. If the highlighted application is a normal, non-merged Ecoin application, then the application is loaded and the user is taken to the Application Editor interface for creating normal applications. If, however, the application that the user has selected is a merger application, then the merger application is loaded from the RDF specification and the user is ushered to the linker.aspx page for defining merger relationships discussed in section 4.2. The user can then edit and modify the merger application starting at this page.

4) Finally, if the Ecoin application engineer wishes to create a *new merger application*, he must select the "CLAMP" button that is circled in item 4. He is then taken to the CLAMP.aspx page described below in figure 4-3.

CLAMP

Context Linking and Merging Process

Pick Which Applicaitons You Want to Merge

2 - TASC Financial Example
395 - Disaster Relief
401 - CIA Factbook
512 - Airfare
513 - Car Rental
514 - Weather
516 - stock
517 - Travel

2 - TASC Financial Example
395 - Disaster Relief
401 - CIA Factbook
512 - Airfare
513 - Car Rental
514 - Weather
516 - stock
517 - Travel

1

Enter name for merged application:

URI . Name

2

Start Merging

3

Figure 4-3: CLAMP.aspx for loading new merger applications.

The CLAMP.aspx page shown in figure 4-3 is used to create *new merger Ecoin applications*. It is used as follows:

1) The user selects two (different) applications to merge from the boxes highlighted in item 1.

2) As with new normal applications, the user must provide the installation URI and application name for the merger application he wishes to create.

3) Once steps 1 and 2 are complete, the user clicks the "Start Merging" button to create a new merger application and begin the linking process.

Upon creating a new merger application, the user has accomplished the first goal of the virtual merging process – creating access to sources from both Ecoin applications. Once the application engineer generates the prolog merger axioms, he will be able to query both sets of sources without any further modifications to the merger application. Of

course, the user may want to create new equivalence relationships to facilitate cross-fertilization or new application elements for the merger application. These are described next, with the assumption that the user has chosen to merge the airfare and car rental applications from CLAMP.aspx above.

4.2. Merging Ontologies

Once the application chooses to create a new merger application through the CLAMP.aspx page or loads a merger application from the TextInterface.aspx page, he is taken to the linker.aspx page that allows him to create or modify equivalence relationships between the ontological elements in the child applications. We now describe how each of these ontological equivalences are declared through the linker.aspx interface.

4.2.1. Virtual Semantic Type Relationships

Figure 4-4 shows the menu for creating equivalence relationships between semantic types in the child applications.
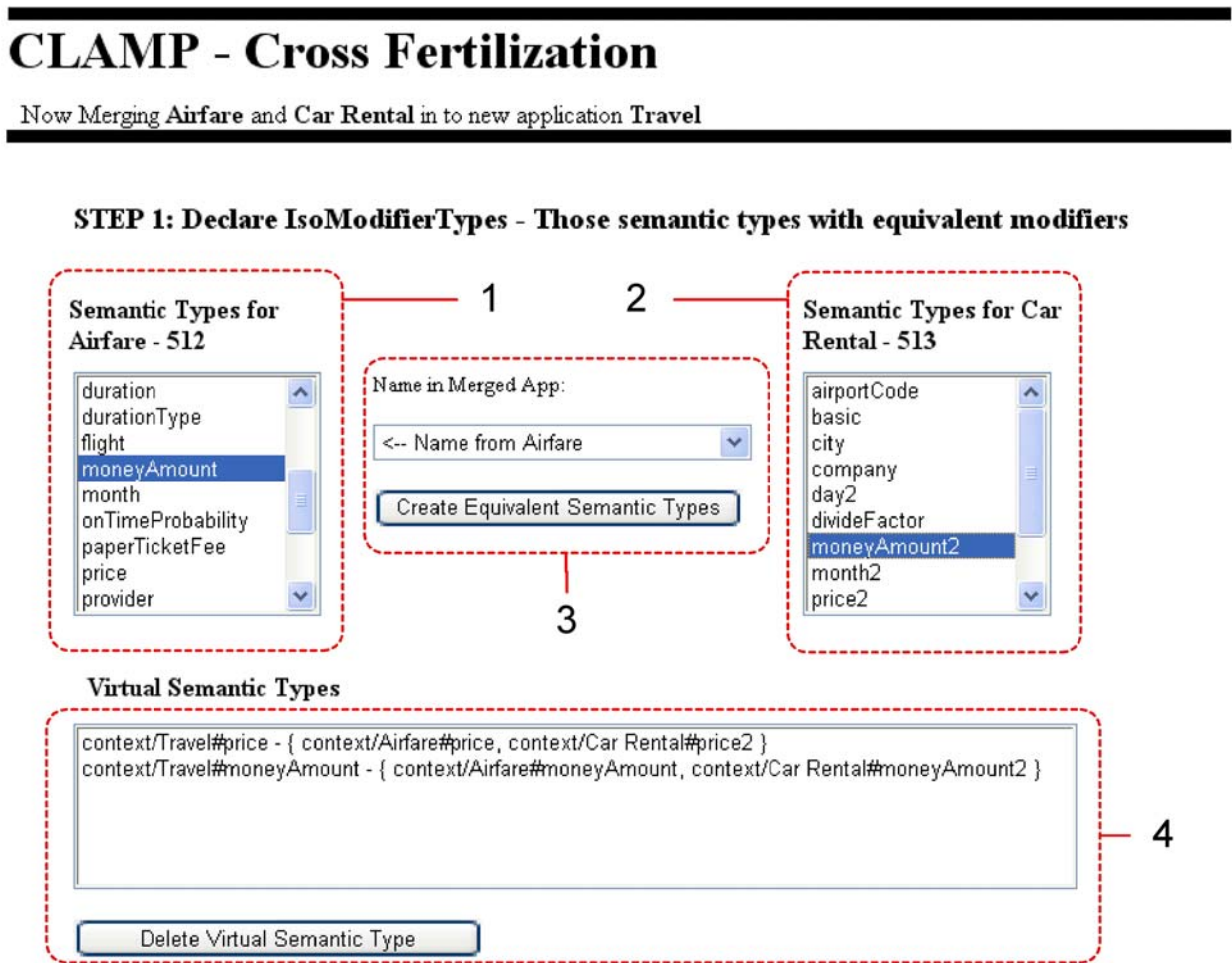

Figure 4-4. The section of linker.aspx that facilitates the definition of IsoModifierType relationships.

1) Item 1 displays all of the logical semantic types contained in the first child application (in this case, the airfare application). The user can scroll down and select a semantic type that will be part of an isomodifierType relationshp.

2) Item 2 displays the same information as item 1, except for the second child application (in this case, the car rental application).

3) Item 3 contains two things of note. First, the drop down box allows the user to select the name that will be inherited up in the merger application for the semantic types that are declared equivalent. This will then specify what the combined semantic types will be referred to as. Once the Ecoin application engineer has selected the semantic types from the child applications that he wishes to equate, and he has selected the name that will be inherited up, the engineer clicks the "Create Equivalent Semantic Types" to create the relationship.

4) The textbox in item 4 displays all of the equivalence relationships that have been declared. For example, it shows "context/Travel#moneyAmount", which is the virtual semantic type representing the equivalence between the "context/Airfare#moneyAmount" and "context/Car Rental#moneyAmount2" semantic types. The user can select one of these relationships and click on the "Delete Virtual Semantic Type" button to delete the relationship if he wishes.

4.2.2.   Virtual Modifier Relationships

The linker.aspx page also allows the Ecoin application engineer to define equivalence relationships between the two other components of an Ecoin ontology: modifiers and attributes. Figure 4-5 shows the menus for defining isomodifer relationships – creating an equivalence between two modifiers whose domain semantic types have been declared as equivalent.

STEP 2: For each new isomodifier type pair declared, define any new virtual attributes and virtual modifiers
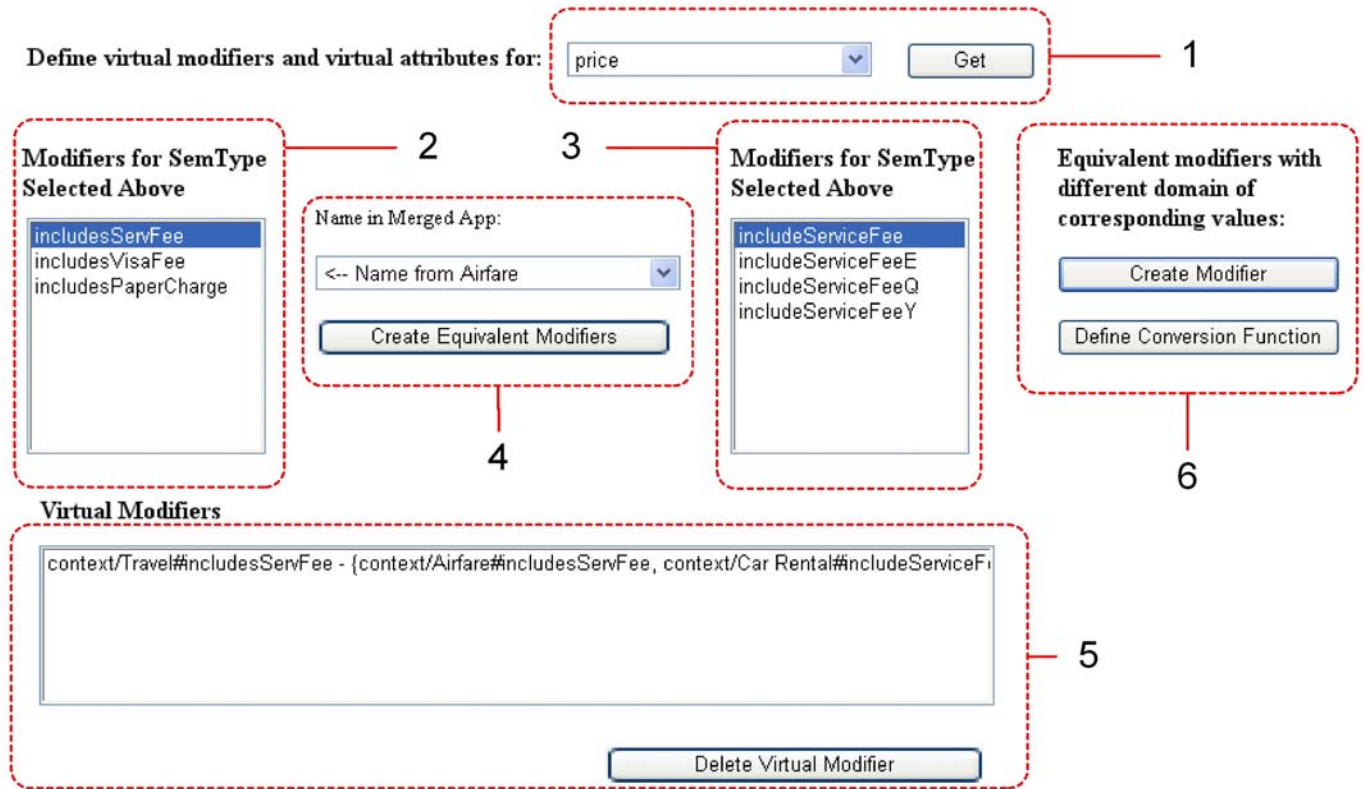
Define virtual modifiers and virtual attributes for: | price ▼ | | Get | — 1

Modifiers for SemType Selected Above
2   3
Modifiers for SemType Selected Above
Equivalent modifiers with different domain of corresponding values:

includesServFee
includesVisaFee
includesPaperCharge

Name in Merged App:

<-- Name from Airfare ▼

Create Equivalent Modifiers

includeServiceFee
includeServiceFeeE
includeServiceFeeQ
includeServiceFeeY

| Create Modifier |
| Define Conversion Function |

4

6

Virtual Modifiers

context/Travel#includesServFee - {context/Airfare#includesServFee, context/Car Rental#includeServiceF

5

Delete Virtual Modifier

Figure 4-5: The section of linker.aspx that facilitates the definition of isomodifier relationships.

1) Before specifying isomodifiers and isoattributes, the user must first specify which virtual semantic type he wishes to extend. Once he selects the virtual semantic type in item 1 and clicks the "Get" button, the CLAMP tool fetches the modifiers for the two isomodiferTypes declared as equivalent for the selected virtual semantic type.

2) The user can then select a modifier from item 2 to declare as equivalent. The modifiers are from the semantic type corresponding to the application displayed on the left side of the linker.aspx page.

3) Similarly, the Ecoin application engineer can select a modifier to declare as equivalent from the second child semantic type from the application that is displayed on the right side of the linker.aspx page.

4) Upon selecting which modifiers will be equivalent, the application engineer chooses the name to be inherited up in the merger application and then clicks the "Create Equivalent Modifiers" button.

5) When equivalent modifiers are created, they show up in the textbox in item 5. Similar to the format for virtual semantic types, this textbox displays the virtual modifiers that have been created, and the child modifiers that are now equivalent. The Ecoin application engineer can select one of the virtual modifiers from the list in item 5 and click the "Delete Virtual Modifier" button which will delete a virtual modifier relationship that was created.

6) When modifiers are defined as equivalent, the Ecoin engine uses the domain of modifier values from both child applications to perform conversion functions. On occasion, the modifier domains in the child applications may be equivalent semantically, but the values themselves are represented differently. For example, consider if two modifiers, context/Airfare#currency and context/Car Rental#currency were defined as equivalent. The Airfare#currency modifier has a domain of {"USD", "GBp"}, while the Car Rental#currency modifier has a domain of {"USDollar", "Pounds"}. These are indeed equivalent modifiers with equivalent modifier values *semantically*, but the representations of their modifier values are not the same. Such a situation requires the Ecoin application engineer to define a new modifier and create a conversion function that will convert the domain of values in one application to another. This can be done by first creating the modifier by clicking the "Create Modifier" button and then clicking the "Define Conversion Function" button in item 5. These buttons guide the user to the standard Application editor interface for creating modifiers and conversion functions [2].

4.2.3.   Virtual Attribute Relationships

With the exception of item 6 above, the process of defining virtual attributes is the same as that for defining virtual modifiers. Figure 4-6 below shows the menus for creating virtual attributes, and the reader should extrapolate their navigation from the discussion on isomodifiers.

At the bottom of the linker.aspx page is a button to move on to the next phase of merging Ecoin applications – defining context relationships. Once you have finished merging the Ontologies for the child applications, click this button and it will take you to the mergingcontext.aspx page, which we discuss next.

Figure 4-6: The section of linker.aspx that allows you to define isoattribute relationships and also proceed to the next phase of merging.

4.3. Merging Contextual Information

4.3.1. Virtual Context Relationships

The mergingcontext.aspx page facilitates linking the contexts and declaring implicit modifier values for the merger application. The process of creating equivalent contexts is guided by the menu shown in figure 4-7.

1) Item 1 lists the contexts from the first child application (airfare in this case). The Ecoin application engineer can select one context from this list to create an equivalence relationship with.

2) Item 2 lists the contexts in the second child application which the application engineer selects to define an equivalence relationship.

3) When the application engineer has selected two contexts to define as isocontexts in the merger application, he then selects which context name to inherit in the

merger application. Once the child contexts and the name to be inherited have been selected, the application engineer clicks the "Create Equivalent Contexts" button to create the isocontext relationship.

Figure 4-7: The section of mergingcontext.aspx that allows the Ecoin application engineer to define isocontext relationships.

4) Once isocontext relationships have been defined, they show up in the format introduced earlier in the textbox in item 4. It lists the newly created virtual contexts (that represent the equivalences between contexts) and the contexts that have been defined as equivalent. In the example above, the c_yahoo context and the c_yahoocar contexts have been declared as equivalent. If the user wishes, he can delete existing isocontext relationships by clicking the "Delete Virtual Semantic Type" button.

4.3.2.   Assigning Values for Implicit Modifiers

The mergingcontext.aspx page also allows the Ecoin application engineer to bind values for implicit modifiers. Recall from chapter 2 that implicit modifiers are those modifiers that aren't explicitly created in an application, but exist implicitly for a semantic type. They must be dealt with explicitly when the application is merged with another whose corresponding semantic type has that modifier explicitly defined. Once the implicit values for a modifier have been defined, the mediation capabilities for that modifier can be used for data in the application that formerly only included the modifier implicitly.

An example will help clarify this point. Refer to figure 4-8 as we discuss the process of defining implicit modifiers and what this definition signifies.
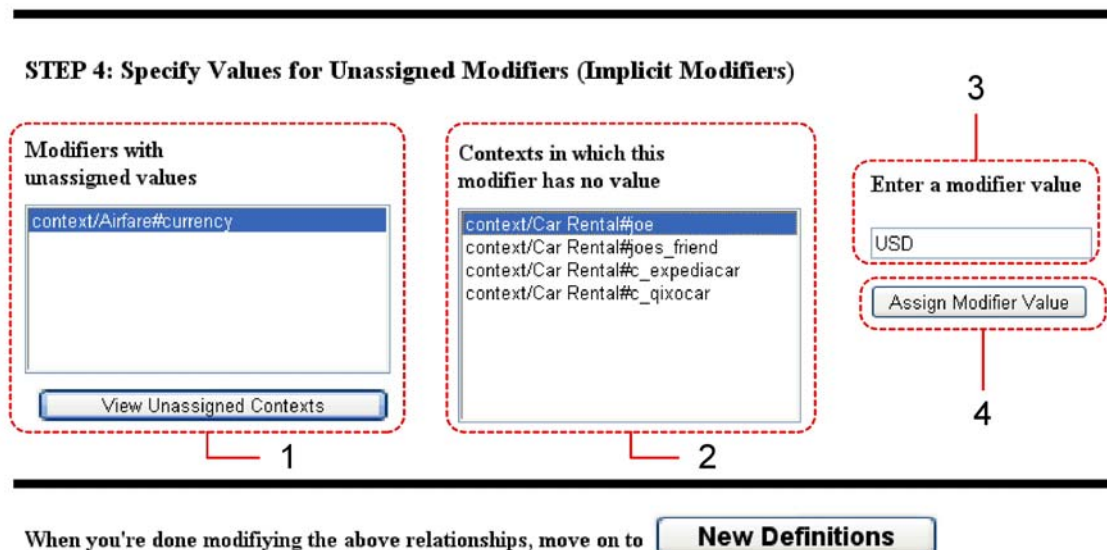

Figure 4-8: The section of mergingcontext.aspx that facilitates the definition of implicit modifier values.

1) The textbox in item 1 displays modifiers inherited by the merger application that do not have values for some contexts defined. In this case, the currency modifier for the moneyAmount semantic types that were declared equivalent does not have modifier values assigned in the contexts from the Car Rental application because the currency modifier was implicit for the moneyAmount semantic type in Car Rental. By selecting a modifier and clicking "View Unassigned Contexts", the CLAMP interface displays contexts in which the modifier requires values to be assigned.

2) The Ecoin application engineer then selects a context to assign a value in item 2.

3) Once a context has been selected, the user can specify the value to assign to that modifier in item 3.

4) Finally, the user can select the "Assign Modifier Value" button to create the implicit modifier relationship. This assigns a value (specified in item 3) to the modifer in item 1 for the unassigned context in item 2. In our case, each of the contexts in the Car Rental application are assigned the value "USD" (meaning U.S. Dollars) for the currency modifier.

4.4. Defining New Application Elements

Once the user has finished defining equivalent contexts and implicit modifier values, he can proceed to defining new Ecoin application elements. This is done by selecting the "New Definitions" button in figure 4-9. Upon clicking this button, the user is taken to the normal Application Editor interface for an Ecoin application, documented by Lee [2].

The Application Editor interface can also be used to modify the virtual semantic types that have been created through merging even further. New modifiers and attributes can be defined. For example, suppose two semantic types, Airfaire#moneyAmount and CarRental#moneyAmount, are linked using CLAMP. Neither of these semantic types had a notion of currency (all moneyAmount values in the Airfare application were in dollars, while the CarRental dealt with all moneyAmount values in Pounds). However, there is a need to have the notion of a currency modifier in the new application. This can be achieved by declaring the two moneyAmount semantic types as equivalent, and then creating a modifier in the Application Editor section for the newly created virtual semantic type that equates the two moneyAmount semantic types.

4.5. Generating the Prolog and RDF Code for Your Application

For merger applications, we are ultimately interested in the Prolog code that is generated so that we can provide the axioms to the Ecoin engine that will then be able to mediate the merger application's sources (both inherited and new) appropriately.

These axioms, as well as the RDF representation of the merger application, are created when the user selects the "Generate" button in the Application Editor after clicking the "New Definitions" button in mergingcontext.aspx. Figure 4-9 displays the resulting RDF code that is generated. This RDF representation can now be used subsequently by the CLAMP tool to load the merger application. Generating the RDF code essentially corresponds to a save operation, where a permanent representation of the merger application is created on disk and can be accessed and modified for later use.

Figure 4-9: : The RDF generated to represent merger application.

Finally, figure 4-10 displays the prolog code that is generated for the merger application. Note that it includes the merger axioms that define the iso-relationships and implicit modifiers that the Ecoin application engineer defined through the CLAMP use interface.

Figure 4-10: The prolog axioms generated by the CLAMP tool to represent the merger application

Chapter 5. Guidelines for Building Ecoin Ontologies

We now turn our attention to one critical part of the Ecoin application: the Ecoin ontology. Our goal in this chapter is to put forth some guidelines that Ecoin application engineers can use to create ontologies for their Ecoin applications. As we shall see, building an ontology with some important considerations in mind can help to make the ontology easier to work with in the future for maintenance, sharing, or even merging.

The Ecoin ontology provides the fundamental mechanisms whereby the Ecoin system can perform context mediation. The ontology provides a conceptualization of the domain of interest using semantic types to represent objects and attributes to represent the relationships between those objects. Furthermore, Ecoin ontologies have a special notion that other ontologies (such as those used in knowledge representation domains) do not: modifiers that encapsulate the notion of context.

Modifiers can be thought of as specialized attributes for semantic types. Let's look at a snippet of an Ecoin ontology to see what a modifier exactly does. Figure 5-1 shows the "moneyAmount" semantic type that represents money in an ontology and the "currency" modifier that is used to encapsulate context information about the currency of the money data.

Context	Value
cxt_yahoo	"USD"
cxt_blmberg	"GBP"
cxt_office_uk	"GBP"
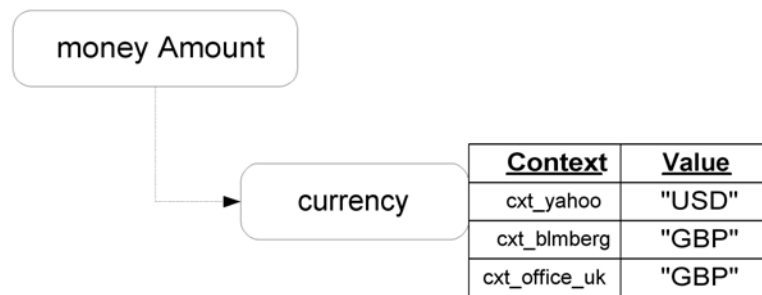
Figure 5-1: How modifiers encapsulate context.

Conceptually, each modifier has a table of context-value pairs that specify the value of the modifier in different contexts. When the Ecoin system knows the source or user context and also the semantic type elevation of a particular piece of data, it can then use this information in the modifier to make the appropriate conversions. For example, lets say that we get a stock price of "25.60" for the Microsoft Corporation from the Yahoo Finance website. This data is elevated to the moneyAmount semantic type and the context for the source is cxt_yahoo, where values are in U.S. dollars. If the user context that will read the data is cxt_office_uk, where the modifier value is British pounds, then the Ecoin system will convert the value "25.60" to British pounds.

With this basic understanding of the Ecoin ontology's role in facilitating context mediation, we can now move on to exploring some principles for creating Ecoin ontologies. We first look at some other research in the area of conceptual modeling and guidelines for building conceptual models. This research will help us focus on some

things that are important when building conceptual models such as ontologies. The other pre-requisite for beginning a discussion on principles and guidelines for making ontologies is to understand how Ecoin applications are used and the contextual requirements of these different situations. We then proceed to elaborating on some basic principles that will make the use and maintenance of Ecoin ontologies easier. We end with a discussion on how one goes about creating the ontology following the principles offered using a sample set of data.

5.1. Related Work

The discussions below help to clarify some of the current research on conceptual modeling and guidelines for building conceptual models.

5.1.1.   Ecoin Research

The current Ecoin research does have some guidelines and principles for creating Ecoin applications. In particular, Kaleem [3] gives the following basic methodology for creating an Ecoin application:

1. Sources: figure out which ones to use, the context issues between them, and wrap them with Chameleon if necessary.

2. Ontology: Develop an ontology that describes the domain in question.

3. Contexts: define the contexts that will exist among different users and sources. Incorporate any context differences in to the ontology by designating values for modifiers.

4. Conversion functions: specify conversion functions to convert data from one context to another.

5. Formalize all of the above in the Ecoin format – specify above relationships using the TextEditor interface or through manually created files for Ecoin.

However, this methodology leaves many questions open, especially about the design of the ontology. How does the design of the ontology depend on the sources and contexts that exist? What are other key principles to follow when creating the ontology? In this chapter, we attempt to answer these questions and make the process of ontology creation clearer.

5.1.2.   Other Ontology Research

There is a great deal of research on using ontologies to represent and manage knowledge. The On-To-Knowledge system developed by Sure [6] represents one such effort. The goals of this system is to use ontologies to represent knowledge in a particular

domain so that tasks such as searching or browsing through information are made easier and more efficient. This differs from the Ecoin system in that ontologies are used for context mediation of knowledge rather than for searching through or managing large amounts of data. Sure also presents a methodology for creating the ontologies used by the On-To-Knowledge system [6]. This methodology includes the principles of analyzing the data and the domain that one will be representing in order to create an ontology that performs optimally for that data, and also cycling through the design process of the ontology to improve it iteration by iteration. We shall adapt these basic approaches to the way that Ecoin ontologies are build.

The Protégé project at Stanford is another effort to facilitate the creation of knowledge management systems [7]. The goal of the Protégé is to provide a tool that knowledge engineers can use to build knowledge management systems. Thus the system exists mainly to facilitate the creation of knowledge bases.

The Protégé project has also developed a basic set of guidelines for developing ontologies [8]. These guidelines are specifically geared towards building ontologies for knowledge management systems that perform tasks similar to those of the On-To-Knowledge system: sharing and querying knowledge. Again, this differs from the ultimate goal of the Ecoin system, which is to perform context mediation of data. Some of the guidelines provided by the Protégé project are nevertheless useful for Ecoin ontologies as well. Firstly, they describe ontology engineering as "necessarily an iterative process" where you create an ontology, see how well it works for your particular application, and then go back and make changes as necessary. Furthermore, they also put emphasis on understanding the domain and scope of your ontology and determining what types of questions your ontology must answer [8].

Gruber also presents a set of design principles for ontologies that are used in knowledge sharing and interoperation of programs based on a shared conceptualization [9]. Some of the relevant principles he puts forth include having clarity in the terms and names defined in the ontology and also creating the ontology so it is extendible in the future. Gruber also provides a case study of building ontologies in an iterative process using the principles that he presents.

There is a great deal of research on ontologies and ontology guidelines, some of which has been presented above. The key distinction between this research and the work that is outlined later is that our discussion focuses around the Ecoin ontology, which has a purpose and structure distinct from traditional ontologies used for knowledge management. However, many of the principles developed in the knowledge management domain will prove useful in developing Ecoin ontologies as well.

5.1.3. Entity Relationship Diagrams

Another important area of conceptual modeling is that of entity-relationship models for designing databases. Like ontologies, entity-relationship models also represent the

objects and the relationships that exist in a particular domain. However, the goal of an entity-relationship model is to serve as a tool for database design [10]. Once an entity-relationship model has been created for a domain, methods exist to transform that model in to a database design [11].

The goal of entity-relationship diagrams are to help build "good" databases – databases that have certain qualities that make them efficient in terms of data storage and performance. Consequently, many of the qualities that make database designs "good", such as normalization and minimal redundancy, do not directly translate to guidelines that are useful for building Ecoin ontologies [11].

However, much of the research surrounding what makes a good entity-relationship diagram on the basis of usability and maintainability are very helpful in our own discussion of what is a good Ecoin ontology. Genero presents qualities such as understandability, simplicity, and modifiability as important characteristics affecting the quality of the design of a conceptual entity-relationship model [12]. Further, he puts forth a different metrics for determining the complexity of a particular model, with each metric measuring the relative number of entities and relationships that exist in a model. He outlines an experiment where database designers are used to provide input as to the complexity of different entity-relationship models. His experiment concludes that one metric that can predict the level of complexity in a particular diagram is the RvsE metric that measures number of relationships as a proportion of the entire entity relationship graph [12].

5.1.4. Software Modeling and UML Diagrams

Many of the widely known principles for software development and engineering are also relevant to the design of ontologies. UML provides one set of design methods to guide software engineers in the development of software systems [13]. Some important ideas that we will borrow from include the notion of modularity in software systems, inheritence in object oriented modeling, and bulding systems based on the scenarios in which a software system will be used.

5.2. The Company Information Example

In chapter 1, we mentioned that mediating data about company information is one domain where the Ecoin system could be used. Recall that Foo Holdings Incorporated (FH Inc.) is looking to streamline its stock trading business. It has branches in the UK and the US and would like to create an application that will allow them to query company information (including stock prices, and basic company data) from several websites online, including bloomberg and yahoo finance. Realizing that web-based querying for company data isn't going to give them the most up-to-date information about prices on the market, they have also started an in-house project to create a data source for the same information that has a direct connection to the market and would be much faster than the web site querying.

Our task in setting up their system is to give them access to the web based data sources as well as any future data sources that are added. Data should be presented to the various offices in a format that reflects their basic assumptions about the data. If there is a conflict between how an office in one of FH Inc.'s branches expects data and how it is represented in a data source, the Ecoin system should perform context mediation to eliminate the conflict.

The sources that we currently will be accessing (along with the data columns of interest) are as follows:

1. finance.yahoo.com (http://finance.yahoo.com/q/pr?s=MSFT) for all U.S. based stocks.
   - Company Name
   - Company ticker symbol
   - Last Trade – last trading price per share of stock in dollars.
   - Trade time – time in Eastern Standard Time.
   - Change – price change of company stock from previous close.
   - Prev Close – the price of the stock at the previous day's close.
   - Day High – the high price of the stock today.
   - Day Low - the low price of the stock today
   - 52 Week High – the high price of the stock in the past year.
   - 52 Week Low – the low price of the stock in the past year.
   - Volume – number of shares of the company stock traded (in units of shares).
   - Market Capitalization – the market capitalization of the company in billions of dollars.
   - Price/Earning – price to earning ratio of the stock.
   - Dividends – the amount paid out per share of stock.
   - Yield – amount paid out as a percentage of the value of the stock
   - Business summary.
   - Financial summary
   - Revenue – the annual revenues of the company.
   - Revenue Growth – the growth in company revenues over the past year.

2. Bloomberg (http://quote.bloomberg.com/apps/quote?ticker=BAY:LN) for all stocks traded in London.
   - Company name
   - Ticker symbol
   - Trading Volume – number of shares of stock traded.
   - Quote time – date and time of the quote in Eastern Standard Time.
   - Stock price – using for british companies, so currency will be in pence.
   - Change – price change of company stock from previous close.
   - 52 Week high - using for british companies, so currency will be in pence.
   - 52 week low. - using for british companies, so currency will be in pence.
   - Market Capitalization – in millions of British pence.
   - Earnings – the company earnings per share of stock.
   - Company information.

- Company news.

3. In addition to these sources, we will have the in-house database, called FHDB, that will be added in a matter of a few months. This database will be very similar to the Yahoo Finance data source, with the following additions and changes:
    - Chief Executive Officer
    - Chief Financial Officer
    - Trading volume will be reported in thousands of shares rather than in units of shares.

The users of this system will be employees in the U.S. and U.K. offices. We will use parts of this example to discuss our principles for building Ecoin ontologies and then we tackle the entire scenario at the end of the chapter.

5.3. Principles for Creating Ecoin Ontologies

Based on the research literature discussed, as well as specific knowledge of how Ecoin ontologies will be used to facilitate context mediation, there are some principles that Ecoin engineers can follow in order to build Ecoin ontologies that best serve their desired purposes.

5.3.1. Creating the Ontology to Serve Its Purpose

The first basic principle is that of understanding the sources and contexts that will exist for the Ecoin ontology being built. Just as many of the principles guiding the design of entity-relationship diagrams and software models require an analysis of the use of the system being built, an analysis of how the Ecoin system is going to be used is key to building a good ontology. The goal of this analysis is to help in building an ontology that is in many ways *complete*. This means that the ontology has a way of mapping to all the relevant data that will be accessed and has mechanisms in place to perform context mediation where it is necessary.

There are two extremes in which the Ecoin system can be used. The first extreme can be considered a very data-centric approach, where there are many data sources and a user needs to get a consistent view of data across all of these sources. Figure 5-2 illustrates this case.
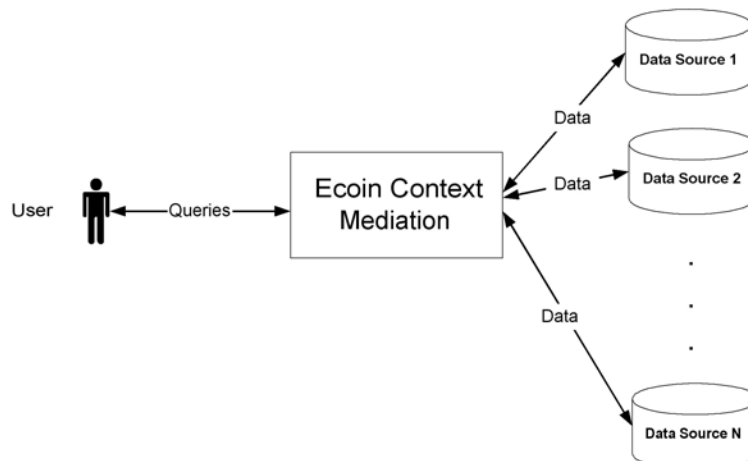

Figure 5-2: Single user accessing many data sources.

70

From our company information example, such a situation can correspond to a scenario where there is only one office querying the data and many sources for company information. In the data-centric case, the ontology needs to include mechanisms (through the modifiers) to encapsulate all of the data source contexts as well as the single user context. For semantic types that are mapped to data columns that have underlying assumptions different from the user, modifiers must be defined to highlight these contextual differences. Furthermore, the Ecoin application engineer must create conversion functions to mediate between these anticipated contextual conflicts. The ontology should include semantic types that map to data columns in all of the data sources.

At the other extreme, there exists the case where there are many users of the Ecoin system who want their own individualized view of data from a single data source. Figure 5-3 illustrates this situation.
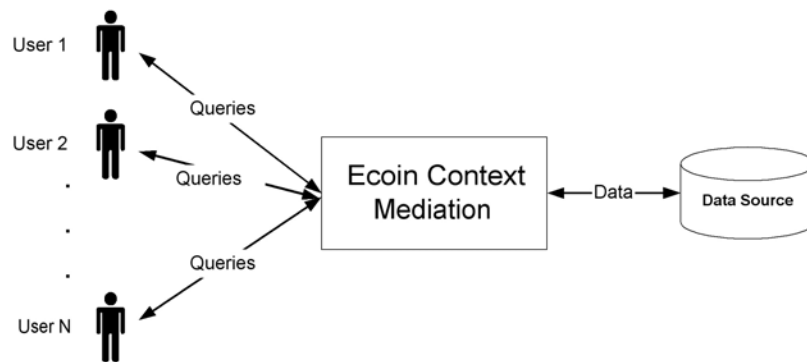
Figure 5-3: Many users accessing a single data source.

In this case, there are many users with their own individual contexts accessing a single data source. An analogue in the company information example is if FH Inc. is only using one source of data, but has many different offices with their own unique ways of interpreting data (such as money values) across the world. The contexts that must be analyzed in this case are the many user contexts as well as the single data source context. The ontology itself must encapsulate all of the data columns in the single data source.

Many applications, such as our FH Inc. example, may be a combination of these two extreme cases. In those situations, it is necessary to consider contexts from all of the users and all of the data sources, and to include semantic types that represent data columns from all of the data sources.

One mechanism that may help in creating complete ontologies is to use a table that describes the data source columns and contextual conflicts. The basic structure of such a table is shown in figure 5-4.
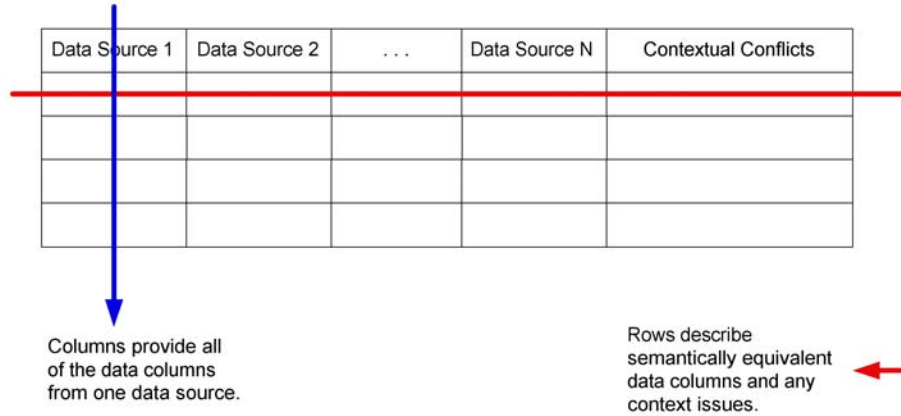
Figure 5-4: A table to help capture all of the semantic types and modifier relationships needed.

Each column in the table describes a different data source. The rows are aligned so that two data columns that represent semantically equivalent data are in the same row. For example, if two data sources both have data values corresponding to the stock price of a company, then those would be in the same row. The last column of a row specifies any contextual conflicts that exist between the data columns in a row of semantically equivalent columns. The Ecoin application engineer can also make note of any contextual conflicts as a result of user contexts in this last column as well.

Such a table helps organize the information that the ontology must necessarily include in order to facilitate all of the context mediation capabilities needed. Each row helps describe what semantic types are necessary. Furthermore, the last column of a row helps to identify where modifiers will be needed in order to perform context mediation. This table may not describe all of the semantic types in the final ontology, because some may be added to improve modularity and clarity in the ontology. However, it provides a good starting point from which the Ecoin application engineer can iterate and improve the ontology as necessary. We shall build such a table for the company information example in section 5.4 and use it to aid in defining the semantic types and modifiers for our ontology.

5.3.2.   Modularity

The next key principle that is important to consider when creating your Ecoin ontology is that of modularity. Modularity is a basic tenet of object oriented programming, and indeed most systems architecture theory. Modular systems tend to be more robust, easier to build, and easier to update. For Ecoin, we must first describe what

modularity means and then show why it is good for building Ecoin ontologies in a modular fashion.

In the context of Ecoin, modularity describes how self-contained subparts of the ontology are. In an Ecoin ontology, semantic types may be linked to one another through
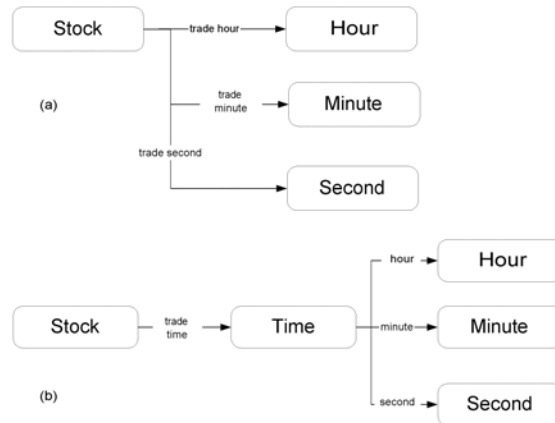
Figure 5-5: A non-modular (a) and modular (b) representation of the trade time of a stock.

attributes that describe how two semantic types are interrelated. The more attributes a semantic type has flowing out of it, the more dependencies it has in the ontology. This greater number of cross ontology dependencies can prevent subparts of the ontology from being self-contained. Figure 5-5 gives an example of coupling between semantic types and how to eliminate it.

In figure 5-6(a) we have a representation of the trade time for a stock as direct attributes of the stock in hours minutes and seconds. The Stock semantic type is coupled with all three of these semantic types and a change in the way the trade time is represented involves changing the stock semantic type as well. However, 5-6(b) shows an instance where Stock semantic type is only coupled with the Time semantic type. The notion of time here is encapsulated by a new semantic type called Time. Now, the stock semantic type is only dependant on the Time semantic type and changes in the way time is represented affects the attributes emanating from Time only and not from Stock.

Besides being a more natural way to represent our ontologies and extend or change them as needed, modularity has many other benefits as well. Firstly, adding the intermediate Time semantic type now allows us to put contextual information in the Time semantic type. If we have to deal with time zone conversions, for example, we can add a "Time Zone" modifier to the Time semantic type, rather than having to add it to the Stock semantic type (where it would be confusing and seemingly out of place). Figure 5-6 shows how adding the notion of a time zone is much more natural with the modular design of the ontology.

Figure 5-6: Adding the notion of time zones is much more natural with the modular ontology design.

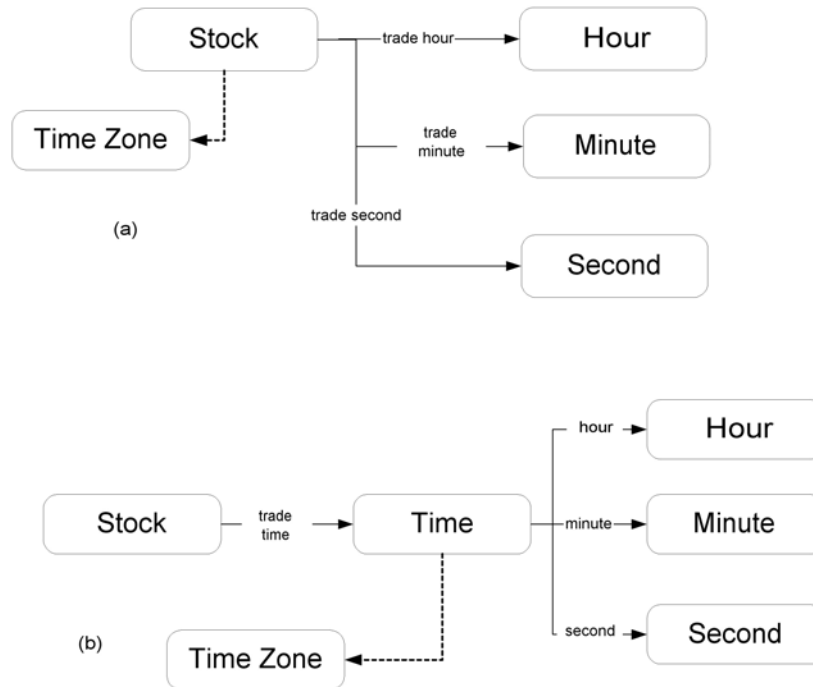Making extension of the ontology more natural isn't the only way that a modular design eases the readability of an ontology. Encapsulating concepts with additional semantic types (as we did with Time above) allows for easier conceptualization of the ontology being designed. Its easier to think of a stock as having a trading time than to think of it having a trading hour, trading minute, and trading second. Furthermore, a modular approach can often lead to fewer overall attributes, making the ontology less complex. Imagine, for example, there were 10 different semantic types that were described by time in a similar fashion to stocks (eg. the time of the previous closing price, the time a stock hit its daily high and low, etc.). For each of these additional semantic types, 3 new attributes would be needed to describe the notion of time using the non-modular approach. However, with the modular approach, only one new attribute per semantic type (pointing to the Time semantic type) needs to be added.

Finally, modularity can also help in sharing and reusing Ecoin ontologies. Suppose that another Ecoin application engineer is building an ontology and needs to have the concept of time as well. He likes the way we deal with time, and so would like to borrow our representation. In the non-modular approach, it isn't clear exactly what he would reuse, since many of the time-related concepts are scattered throughout the ontology, with some concepts attached to non-time related entities (such as the notion of time zone). The modular design, on the other hand, makes it easy to determine how to borrow our representation of time – simply take the Time semantic type and everything outgoing from that semantic type (in terms of modifiers and attributes). This defines a sub-ontology that neatly defines our conceptualization of time.

5.3.3.   Maintainability

Maintainability refers to how easy or difficult it is to modify, reuse, or extend an ontology, particularly by individuals who didn't initially create that ontology. Maintainability is an important principle to apply to Ecoin ontologies because they require a great deal of maintenance. As the nature of data sources changes and as the users using those data sources changes, the ontology must be adapted to reflect any new data columns and context issues that come up. Furthermore, the process of virtually merging two ontologies depends heavily on maintainability. An Ecoin application engineer can not identify isomodifiertype, isomodifier, or isoattribute relationships correctly if he doesn't even understand each individual ontology completely!

There are many factors that go in to describing how maintainable a particular ontology is. As we indicated in the previous section, modularity can help clarify concepts represented in an ontology and thus make it much easier to read. Readability of diagrams is one aspect that we can say safely transfers from the entity-relationship domain to the Ecoin ontology domain. Thus, the findings of the Genero [12] study that say engineers have the easiest time reading diagrams with a minimal number of relationships can be extended to Ecoin ontologies: minimizing attributes will generally help to make the ontology seem simpler. Modularity can help with this process as well.

Avoiding esoteric or misleading names is another important way to increase maintainability. While I doubt that people will create ontologies that have semantic types and attributes with single letter or numerical names, there are many cases where people may name ontology elements in a way that other people can misunderstand. For example, one individual may have a semantic type Time that only refers to the time of day, while another individual uses Time to refer to absolute time (including the date). Such naming conflicts can be alleviated by introducing a certain level of standardization, as we shall discuss in the next section.

5.3.4.   Standardization

The final principle we offer for building Ecoin ontologies is that of standardization. This essentially means following conventions that are accepted or used by a wide community of Ecoin users. For example, one convention could be what the notion of time encapsulates – only time of day or an absolute sense of time. Following this standard can help avoid confusion when others parse through your ontology. Setting up naming conventions for specific concepts will encourage sharing and reuse of ontologies, since it will allow a greater number of people to understand one another's ontologies.

Another area that can be helped by standardization is modifier values. Different Ecoin communities may have different conventions for naming modifier values. As an example, let's go back to the moneyAmount semantic type and its currency modifiers. Lets assume, as in our Airfare and Car Rental example from the discussion on CLAMP, that we are merging two applications with the notion of moneyAmount. This time, however, both applications have the notion of what currency is. Figure 5-7 describes this scenario.
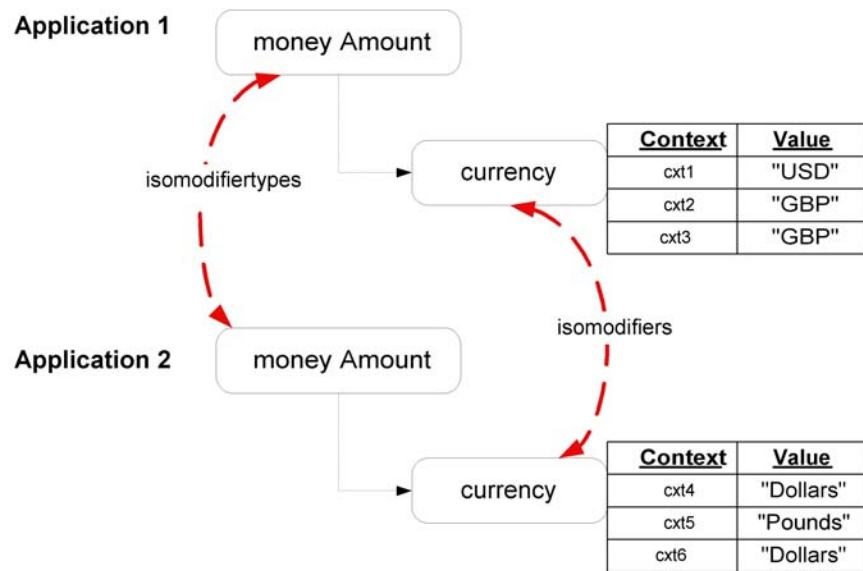
Figure 5-7: The need to have standardized modifier values.

In this case, we want to declare the two moneyAmount semantic types as isomodifiertypes and subsequently the two currency modifiers as isomodifiers. However, the system of modifier values for the two currency modifiers is different – application 1 uses three-letter codes while application 2 uses the currency names. According to Kaleem [3], this requires us to declare an additional modifier in the merged application that specifies which application a particular modifier value is coming from. Using this additional modifier, a conversion function can be defined that converts modifier values from one application to another. Thus, it is possible to reconcile the differences in modifier domain values and use a single conversion function to convert between the combined set of contexts from applications 1 and 2.

But why not just try to avoid having to do this in the first place? It makes for a much simpler virtually merged ontology and a much simpler merging process if we just had the names of the modifier values the same in both applications. In this case, this could have been achieved if the Ecoin application engineers who build each ontology followed a particular convention for naming currencies (which are a very common source of contextual conflict and used in many different applications).

5.4. Creating an Ecoin Ontology for Company Information – A Case Study

We now use our ontology principles to help us build an ontology for the company information application that FH Inc. needs. Figure 5-8 describes the users and the sources that FH is going to be working with. There will be contexts for each of the users and each of the sources that are described above.
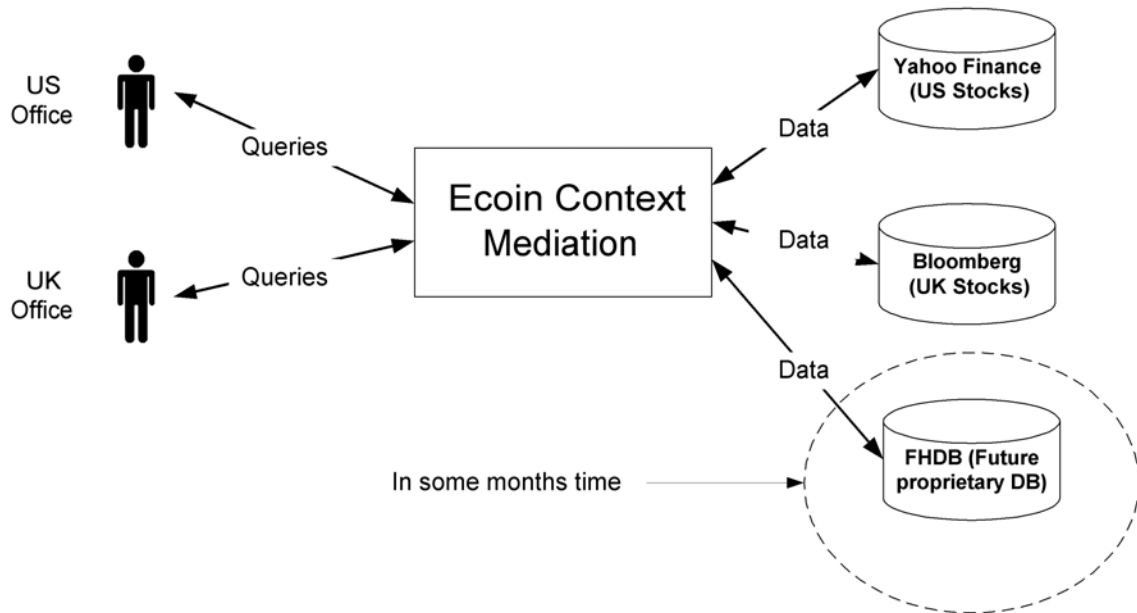
Figure 5-8: The contexts (both user and source) that exist for the company information example.

We now create a table that describes the data columns from each of the data sources, as well as any context issues that exist. The table is shown in figure 5-9. As we discussed in section 5.3.1, we can use this table to help us identify the semantic types that we need to create for our ontology as well as any modifiers that will need to be defined to handle contextual conflicts.

Yahoo Finance	Bloomberg (UK Stocks)	FHDB	Context Issues
Company Name	Company Name	Company Name	
Ticker Symbol	Ticker Symbol	Ticker Symbol	Company ticker symbols from Bloomberg and FHDB have suffixes indicated the stock exchange. Yahoo assumes everything is in the NYSE.
Last Trade	Stock Price	Last Trade	Stock prices from Yahoo and FHDB are in dollars, while those from Bloomberg are in pence (1/100 of a Pound). The UK office expects values in pounds while the US office expects dollars.
Trade Time	Quote Time	Trade Time	All times are currently in Eastern Standard Time. In the future, offices may need to have the times according to their own time zones.
Change	Change	Change	Same currency assumptions as for the last trading prices for the stocks.
Prev Close		Prev Close	Same currency assumptions as for the last trading prices for the stocks.
Day High		Day High	Same currency assumptions as for the last trading prices for the stocks.
Day Low		Day Low	Same currency assumptions as for the last trading prices for the stocks.
52 Week High	52 Week High	52 Week High	Same currency assumptions as for the last trading prices for the stocks.
52 Week Low	52 Week Low	52 Week Low	Same currency assumptions as for the last trading prices for the stocks.
Volume	Trading Volume	Volume	
Market Capitalization	Market Capitalization	Market Capitalization	The Yahoo site and FHDB report these values in billions of dollars, while Bloomberg reports them in millions of British pounds.
Price/Earnings		Price/Earnings	
Dividends	Earnings	Dividends	Same currency assumptions as for the last trading prices for the stocks.
Yield		Yield	
Business Summary	Company Information	Business Summary	
Financial Summary		Financial Summary	
Revenue		Revenue	Same currency assumptions as for the last trading prices for the stocks.
Revenue Growth		Revenue Growth	Same currency assumptions as for the last trading prices for the stocks.
	Company News		
		Chief Financial Officer	
		Chief Executive Officer	

Figure 5-9: A table describing the data columns and contextual conflicts that exist in the company information example.

Note that we consider the data sources that currently exist as well as the FHDB data source that will be integrated in the future. This will allow us to create an ontology that can easily incorporate the new data source once it is ready. We read down the list of data sources and create semantic types accordingly. We also create modifiers for the semantic types that correspond to those data columns that will have some sort of contextual conflict.

The next step is to go through this table and define the semantic types and modifiers that need to exist to describe all of the data values that exist as well as all of the context issues. Figure 5-10 on the next page describes the semantic types and modifiers that are created. Note that in this and all other ontology diagrams, the reader should assume that semantic types inherit from the "basic" semantic type unless otherwise specified. Following the principle of increasing maintainability of our ontology, I've named the semantic types as clearly as possible to reflect the concepts they represent. Furthermore, the currency modifier will have value bindings for different contexts using the three letter code that is used to look up currency exchange rates on most currency conversion websites (eg. http://www.oanda.com/). This is an attempt to standardize the values used for this modifier based on a common website used to convert currencies.

In this first draft of our ontology, there is a currency modifier for all of the semantic types corresponding to data columns that have assumptions about currency on the part of the sources or users. The currentStockPrice, stockPriceChange, dayHigh, dayLow, 52WkHigh, 52WkLow, and prevClose semantic types have a scale factor modifier to deal with the issue that the Bloomberg data source represents stock prices in pence rather than in pounds. The UK office expects these data fields in Pounds and the US office in dollars (where British currency is converted to dollars using the exchange rate between the dollar and the pound, not the dollar and the pence). The marketCapitalization semantic type also has a modifier to express the notion of scaleFactor, where different sources (and users) have different assumptions about the scale factor (ones, millions, billions, etc.) that the market capitalization of a company is expressed. There is also the tickerFormat modifier for the tickerSymbol semantic type to express the context issues that exist between the different sources and users on how the ticker symbol of a company is expressed (as to whether it includes the stock exchange symbol as well). The set of semantic types shown in figure 5-10 cover all of the rows that are presented in figure 5-9 for the various data sources in the company information example.
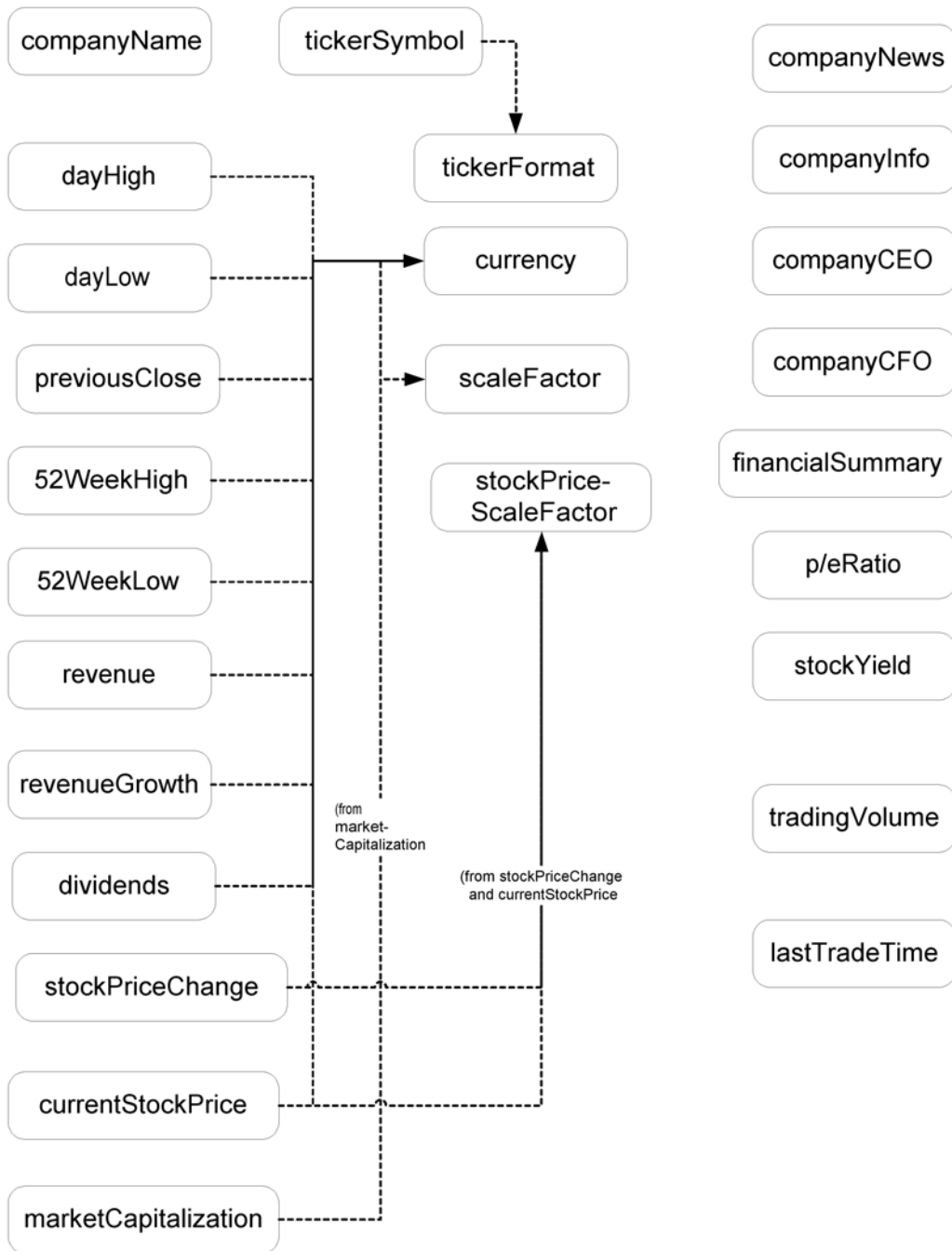
Figure 5-10: A first draft of the semantic types and modifiers for the company information example.

Next, we follow the basic approach of many of the other conceptual modeling approaches and iterate through our ontology to create the attributes and semantic types until we are satisfied with the readability and structure of our ontology. As we iterate and improve the ontology, we keep the principles of modularity, maintainability and standardization in mind.

The first thing that one will note from figure 5-10 is the number of semantic types that exist with a modifier of currency. Let's try to see if there is a better way to represent these semantic types so that this redundancy is more expressed better. We can use the notion of inheritance in this case and create a new moneyAmount semantic type from which all of these "money-related" semantic types inherit. In some respects, this is an instance of modularity since it allows us to encapsulate all of the semantic types with a currency modifier under a single concept – moneyAmount. Then, only the moneyAmount semantic type needs to have the currency modifier defined and all of its descendants inherit this semantic type.
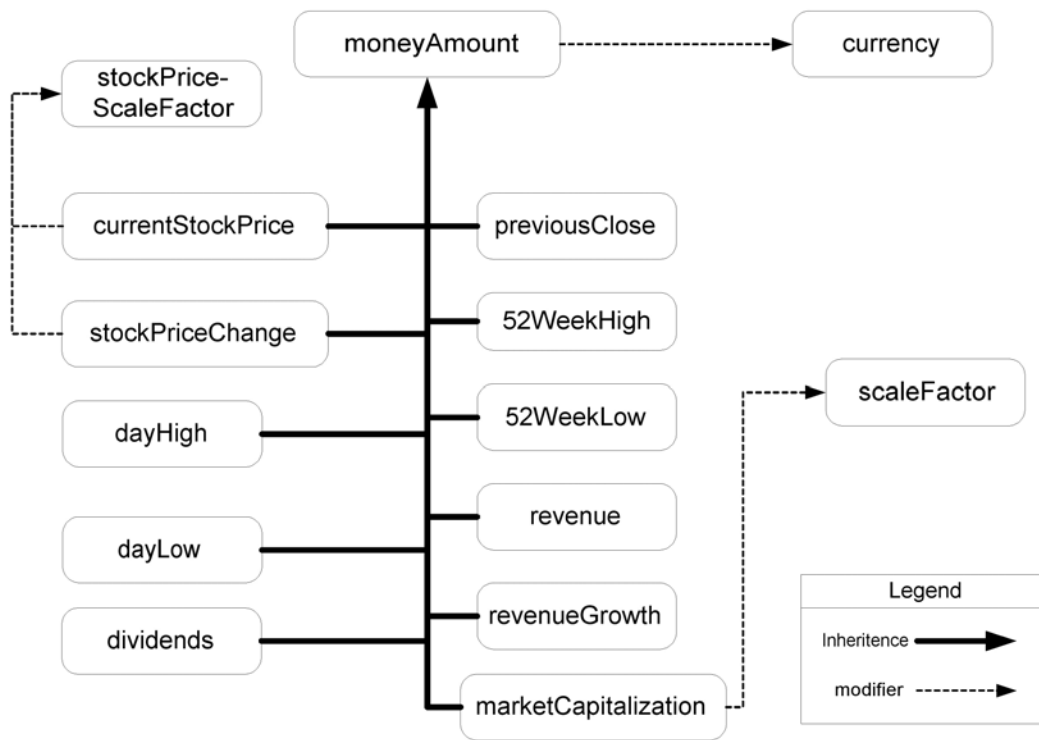
Figure 5-12: Creating the moneyAmount semantic type to encapsulate the notion of currency.

We also want to create two new semantic types to introduce greater modularity in our ontology. The first semantic type encapsulates the notion of a company and so is appropriately named "company". Any other semantic types that directly describe qualities of the company itself will be linked to the company semantic type via an attribute relationship. The other semantic type that we introduce is the notion of a stock,

with a semantic type called "stock". The stock semantic type encapsulates all of the information we have about stocks for a company. The company semantic type will have stock as one of its attributes, since every company has a stock. We also rename the lastTradeTime semantic type and call it just Time. The stock semantic type will have an attribute called lastTradeTime that will point to Time. This is so that we have the Time semantic type and can turn it in to a placeholder to introduce modularity if the way time is represented changes in the future. Right now, we're assuming the last trade time is just a string. However, if in the future the trading time is represented with a date and time of day, we will have to create many attributes of stock to represent the time at which the stock was last traded. With the Time semantic type, we'll just have to add attributes for the Time semantic type to reflect the new way that time is represented.

Figure 5-12 shows a complete ontology with all of the semantic types, modifiers, and attributes that we've described so far. Attributes share the name of the semantic type in their range (at the arrowhead) unless otherwise noted.
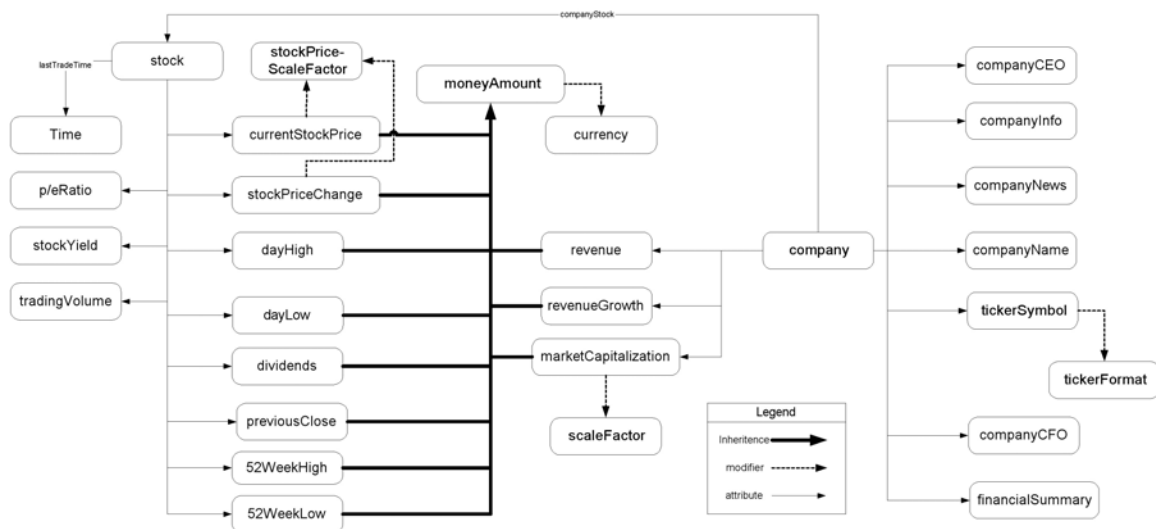

Figure 5-13: A full ontology for the company information example.

We can now iterate through this ontology and change it as we feel appropriate for our specific application. As it stands, the ontology is complete and has an acceptable level of modularity and readability. The amount of modularity that can be introduced is great, as is the amount of information that can be collected together. The extent to which the ontology in 5-12 can be improved is mainly a function of how much modularity and simplicity the Ecoin application engineer requires. We describe the changes that can be made in further iterations below.

1) Many of the semantic types that have been created can actually be collapsed in to individual semantic types. For example, the notions of companyCEO and companyCFO can all be collapsed in to one semantic type called "companyOfficer". Similarly, you can collapse currentStockPrice, dayHigh, dayLow, previousClose, 52WeekHigh, and 52Week low in to one semantic type called "stockPrice". This would eliminate the many individual modifiers emanating from each of these semantic types to represent the scale factor issue with stock

prices. However, if you need to isolate any of the concepts represented by the collapsed semantic types in to future (such as assigning a modifier to one of the collapsed semantic type), you will need to recreate the semantic type, recreate all of the attribute relationships that were necessary for this semantic type, recreate any inheritances that existed, and then define the new modifier. It may just be easier to leave the ontology in a "verbose" state unless you expect that it won't require much modification in the future.

2) Also, a new semantic type called "financialStatus" can encapsulate the company financial information such as revenue, revenueGrowth, marketCapitalization, and financialSummary. This would modularize the part of the ontology that represents all information about the company financial state.

3) Instead of collapsing companyCEO and companyCFO into a semantic type called companyOfficer, we can actually modularize the notion of companyOfficer. We create the companyOfficer semantic type and then keep companyCEO and companyCFO as subclasses of companyOfficer.

As we've learned from our company information example, creating Ecoin ontologies is more of an art than a very algorithmic process. There are definitely some principles that an Ecoin application engineer can follow when creating his ontologies, and we've introduced some of them in this chapter. Beyond understanding the basic contexts and data columns that your ontology must account for, the task of creating ontologies is a process of iterating through the ontology, applying the other principles (modularity, maintainability, standardization) as much as appropriate.

Chapter 6. Conclusion

I now conclude this thesis with a discussion on the contributions that we initially set out to do. As I list each contribution, I shall also summarize the relevant work that was presented. This chapter ends with a discussion of future research directions following the work of this thesis.

6.1.   Contributions

6.1.1.   An Enumeration of the Components of an Ecoin Application

In chapter 2, we discussed the different components of an Ecoin application. In particular, we noted that an Ecoin application consists of the relations, the ontology, the contexts, the elevations and the conversion functions. We described some sample applications using this basic understanding of an ontology.

6.1.2.   The CLAMP Tool for Merging Ecoin Applications

One of the key goals of this thesis was to describe the CLAMP tool. The basic purpose of the CLAMP tool is to facilitate the virtual merging of Ecoin applications. The CLAMP tool provides an interface that guides the Ecoin application engineer through the merger process and generates the necessary prolog axioms to describe the merger application.

In addition to its basic goal of supporting the merging process, there are some other design goals that we wanted to achieve with CLAMP. CLAMP should support merging of Ecoin applications two-at-a-time and should also support the notion of hierarchical merging, where you can merge merger applications. Furthermore, we wanted CLAMP to be integrated with the existing Application Editor infrastructure, especially considering that the Application Editor would be used to extend merged applications with new application definitions.

We achieved the goals of CLAMP by first creating a unique naming system for Ecoin applications and application elements. This was essential so that we could uniquely identify applications and application elements when two separate applications are merged.

With a naming convention in place, we then described the basic information required for merging applications. This information included structures for representing the isomodifiertype, isomodifier, isocontext and isoattribute relationships. We then showed how merger applications are represented in runtime using the Internal Coin Model, in a persistent manner using RDF, and for the Ecoin system as prolog merger axioms.

6.1.3.   A User Manual for the CLAMP Tool

After demonstrating how the CLAMP tool actually represents merger applications and generates prolog axioms, we stepped back and discussed the user interface of CLAMP. The CLAMP tool provides a menu-driven interface that walks the user through the application merging process. Among the important steps in this process are defining which applications are being merged, creating the equivalence relationships between semantic types, modifiers, attributes and contexts, defining any implicit modifier relations, and finally extending the merger application as is appropriate.

6.1.4.   Principles for Building Ecoin Ontologies

Another major objective of this thesis was to present some principles to help Ecoin application engineers in creating ontologies for context mediation. Towards this end, we identified four important principles that Ecoin application engineers can keep in mind while developing their ontologies. These principles are:

1)   Creating the ontology to serve its purpose – this involves understanding how the ontology fits in to the overall goal of the Ecoin system to perform context mediation. With this understanding, it becomes clear that the ontology must be created with consideration for the data values from sources as well as contextual conflicts that exist between sources and users.

2)   Modularity – This principle involves creating the ontology such that we can identify submodules of the ontology that encapsulate particular concepts. This process can help make an ontology easier to understand and also allows extendibility in the future in case new contextual issues come up.

3)   Maintainability – Ecoin ontologies will often be maintained and shared by many people. Therefore, they must be easy to understand, modify, and reuse.

4)   Standardization – this last principle is important because it eases the process of reusing and merging Ecoin ontologies in the future. We showed an example of how a lack of standardization in modifier value domains led to complexities in the merging process.

We then applied our basic design principles towards building an Ecoin ontology for representing company information. The first design principle helped to provide a good starting point for understanding the semantic types and modifiers that are needed in the ontology to deal with the contextual issues that exist in a particular domain. We then build in the attributes and modified the ontology step-by-step following the design principles put forth. We ended with a basic ontology that represents the company information domain and provided further changes that could be made to incorporate as much simplicity and modularity as the user desires.

6.2.   Future Work

The work of this thesis raises many interesting questions and issues that provide excellent research directions for the future. With regards to application merging, we took a big step in the implementation of CLAMP. However, there must be a broader vision of how Ecoin applications are shared and reused among a community of Ecoin application engineers. Such a vision requires having a platform to show and exchange Ecoin applications through a medium such as the world wide web. In this way, Ecoin communities can share their solutions for context mediating particular domains and eventually the bulk of the work in building an Ecoin solution to a context mediation problem can be that of finding the right application somewhere rather than building one yourself.

The CLAMP tool represents a significant extension to an already existing system: the Application Editor. It would be interesting to see how a complete environment for creating and merging Ecoin applications can be built from scratch. One area for improvement in both CLAMP and the Application Editor is incorporating a more graphical user interface rather than a menu-driven interface.

Finally, the ontology creation principles that we discussed are just the tip of the iceberg in researching what makes a good ontology. There are many open questions that can be explored in the future. How should inheritence be dealt with in Ecoin ontologies, and where should modifiers be placed in an inheritance hierarchy? Are there clearer signals to help us decide when to stop introducing more modularity and collapsing semantic types where necessary? Are there other key ontology qualities that are desirable when building Ecoin ontologies? Is there a way to build Ecoin ontologies such that they provide capable context mediation capabilities and can also be used in knowledge application domains?

References

[1] Berners-Lee, Tim. "The Semantic Web". Scientific American. May, 2001.

[2] Lee, Philip. Metadata Representation and Management for Context Mediation. Masters of Engineering thesis, Massachusetts Institute of Technology, Sloan School of Management. May, 2003.

[3] Kaleem, M. Bilal. CLAMP: Application Merging in the Ecoin Context Mediation System Using the Context Linking Approach. Masters of Engineering thesis, Massachusetts Institute of Technology, Sloan School of Management. August, 2003.

[4] Firat, Aykut. Information Integration Using Contextual Knowledge and Ontology Merging. PhD Dissertation, Massachusetts Institute of Technology, Sloan School of Management. August, 2003.

[5] Manola, Frank. RDF Primer, http://www.w3.org/TR/rdf-primer/.

[6] Sure, York et al. On-To-Knowledge Methodology. Institute AIFB, University of Karlsruhe. 2001.

[7] Gennari, John H. et al. The Evolution of Protégé: An Environment for Knowledge-Based Systems Development, http://www.smi.stanford.edu/pubs/SMI_Reports/SMI-2002-0943.pdf.

[8] Noy, Natalya F. Ontology development 101: A Guide to Creating Your First Ontology, http://protege.stanford.edu/publications/ontology_development/ontology101.pdf.

[9] Gruber, Thomas. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. Formal Ontology in Conceptual Analysis and Knowledge Representation. March, 1993.

[10] Chen, Peter Pin-Shan. The Entity-Relationship Model – Toward a Unified View of Data. ACM Transactions on Database Systems, Vol.1, No.1. March, 1976.

[11] Teorey, Toby J. et al. A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model. Computing Surveys, Vol. 18, No. 2. June, 1986.

[12] Genero, Marcela et al. Measuring the Quality of Entity Relationship Diagrams. ER2000 Conference, LNCS 1920. 2000.

[13] Hunt, John. Guide to the Unified Process. Springer: New York, 2003.