

CLAMP:
Application Merging in the ECOIN Context Mediation
System using the Context Linking Approach

M. Bilal Kaleem

Working Paper CISL# 2003-05

August 2003

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

CLAMP:
Application Merging in the ECOIN Context Mediation System using the
Context Linking Approach

by
M Bilal Kaleem

Submitted to the
Department of Electrical Engineering and Computer Science

August 22, 2003

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer [Electrical] Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

Integrating data from heterogeneous data sources means dealing with context differences. That is, differences in the assumptions that are made regarding format and interpretation of the data. The Context Interchange (COIN) group has developed a formalism to describe the context assumptions of data sources and data receivers. An ECOIN application ties various sources together by being mapping them to a common ontology, or domain model. ECOIN applications allow the context differences between data sources to be resolved through context mediation. Users may then desire to merge together multiple ECOIN applications so that context differences across a much wider range of sources can be resolved with minimal additional effort.

Accordingly, the work of this thesis focuses on the problem of merging ECOIN applications. The approach to merging discussed herein is termed context-linking. Context-linking allows the merging of ECOIN applications with minimal effort having to be spent on merging the ontologies of the two applications. This is achieved by employing a virtual approach to ontology merging that gives the appearance of a merged ontology. This thesis describes the merging process, presents a detailed case study, demonstrates the benefits of merging and presents the design for CLAMP, a tool that facilitates ECOIN application merging.

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

Table of Contents

ABSTRACT	2
1. Introduction	5
1.1 – Dora the Explorer	5
1.2 - Multiple Sources mean Context Differences	6
1.3 – The problem of heterogeneous data source integration and the ECOIN approach	7
1.4 - The Application Merging Problem	9
1.4.1 - Virtual Merging versus Materialized Merging	11
1.4.2 - Motivation	11
1.5 - Objectives and Roadmap.....	12
2. The Applications: Airfare, Car Rental and Their Merger	13
2.1 The Airfare Application	13
2.1.1 – Steps to development.....	13
2.1.2 - Sources	13
2.1.3 - Ontology and Elevations.....	14
2.1.4 - Modifiers and Context.....	15
2.1.5 - Sample Queries	16
2.1.6 - Future expansion: accommodated by the ontology or left up to merging?.....	20
2.2 Car Rental Application	21
2.2.1 – Motivation and Context Differences	21
2.2.2 – Sources	22
2.2.3 - Ontology and Elevations.....	22
2.2.4 - Modifiers and Context.....	24
2.2.5 - Sample Query.....	25
2.3 - Merging Airfare and Car Rental	27
2.3.1 - Goals of Merging Two Applications.....	28
3. – Merging Applications.....	30
3.1 – Materialized Merging versus Virtual Merging.....	30
3.1.1 – Materialized Merging	30
3.1.2 - Virtual Merging through Context Linking.....	33
Context Linking – Merging is Driven by Context Differences.....	33
3.1.3 - Linking Contexts Between Airfare and Car Rental	34
3.2 - How to merge applications – The Merging Algorithm and the Merger Axioms.....	36
3.2.1 – Merger Axioms.....	36
3.2.2 – How the Merger Axioms Work.....	37
Upward Inheritance	37
Unique Naming	38
How the Abduction Engine Uses the Axioms.....	38
3.2.3 – The Merging Algorithm	38
3.3 – Understanding the Merger Axioms	40
3.4 – The Merger Axioms of Airfare + Car Rental	41
Seamless access to sources from both applications:.....	41
Cross fertilization of contexts:.....	41
Extending the merged application	43

Ismodifiers, isocontexts and isoattributes:	47
3.5 Sample Run of Merged Application.....	47
3.5.1 – Seamless Access to Sources	47
3.5.2 – Cross Fertilization of Contexts.....	48
3.5.3 – Value Added Benefits.....	49
3.6 – Complete List of Merging Capabilities	51
3.7 – Chapter Summary – ECOIN Application Merging in a Nutshell.....	52
4. CLAMP – A tool to facilitate Application Merging	54
4.1 - What is CLAMP?.....	55
4.2 - What assumptions are made about the user?.....	55
4.3 - The Tool’s Approach	56
4.4 - Design Overview.....	57
4.4.1 – Integrated with Application Editor.....	57
4.4.2 – Architecture	58
4.5 – CLAMP Interface – How the user is led through merging	59
4.5.1 – First page: Seamless Access to Sources	60
4.5.2 – Second page – context reconciliation and cross-fertilization	61
4.5.3 – Extending the merged application	63
4.5.4 – Generating merger axioms	63
4.6 – How Application Editor needs to be modified to support CLAMP	64
4.6.1 – How the Internal COIN Model (ICM) must be changed.....	65
Ont_Isomodifiertype.....	66
4.6.2 – How the API must be changed.....	67
5. Possible improvements/extensions.....	69
5.1 – Graphical Enhancement.....	69
5.2 – Implicit modifier detection.....	69
5.3 – Conversion function assistant.....	69
6 - Related Work	71
7. Conclusions	76
7.1 Summary of thesis and contributions	76
7.2 Evaluation of motivations for thesis.....	76
7.3 – A look ahead.....	78
References	79
Appendix A – Application Prolog file from Airfare	81
Appendix B – Application Prolog file from Car Rental.....	95
Appendix C – Merger Axioms file for Airfare + Car Rental Merger	103
Appendix D – Additions to API to Support CLAMP’s Changes to ICM	106
Appendix E – A Summarized Demo Of Merging.....	107

1. Introduction

Over the last decade, as the cost of data storage has plummeted, access to and use of the Internet has proliferated, and distributed organizations have become much more common, the number of data sources has risen considerably [5]. Furthermore, users not only rely on more data sources but also on increasingly different types of data sources. Yet users still expect to understand and communicate with these heterogeneous sources in terms they understand – making assumptions about the format and semantics of the data that may not be true. To understand this problem better, we begin with an example of a user that is dealing with multiple Internet data sources.

1.1 – Dora the Explorer

Meet Dora. She is quite the world traveler. One place she has not yet been is Istanbul, Turkey. But before she can pack her bags, she must endure perhaps the most tedious part of traveling: finding the cheapest flight. Being the thrifty traveler that she is, she knows to check multiple airfare aggregator websites because while they all claim to check hundreds of airlines for the lowest fares, they usually have different arrangements with various airlines to market their fares in a special manner. This leads to two key types of differences between airfare aggregator websites:

- Different sites may present the same flights but different prices for the same trip
- Different sites may present completely different flights *and* prices for the same trip

For example, Dora sent the same query to Orbitz and Travelocity and got a \$600 difference for their best fares to Istanbul:



1.2 - Multiple Sources mean Context Differences

There are several bothersome issues that Dora faces in searching different sources for airfares. These issues can be divided into two classes:

1. Issues related to juggling multiple sites.

For example, having to run the same query on multiple sites then juggling between them, struggling to remember which site offered what fare while adjusting dates and times of travel.

2. Context differences between the sites.

Multiple sites means differences in the meaning of the results returned from each site (due to the different contexts that each site may assume). Examples of problems due to context differences are:

- What does price include? Taxes? Service fees? For example, Orbitz includes taxes but does not include a service fee until a screen much later in the process. Expedia includes a service fee right from the beginning. So for each site, Dora would have to account for context differences when comparing the prices.

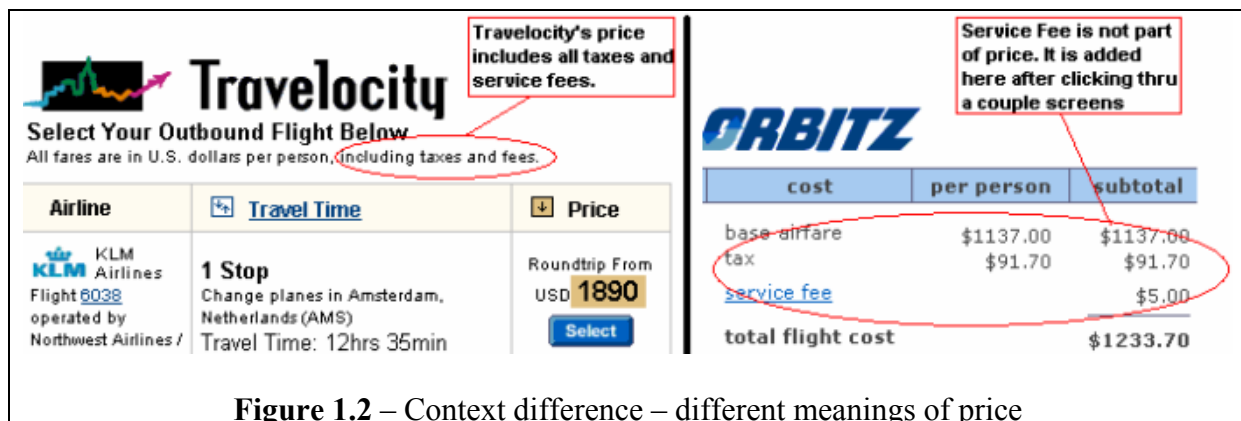


Figure 1.2 – Context difference – different meanings of price

- Aggregators are based in many countries and return fares in different currencies. Dora would need to determine what currency the source uses, what the latest exchange rate is and then convert the price quoted.
- Dora does not like to spend a lot of time on planes so she wants to minimize her flight time. Thus she needs to know whether flight time quoted by a site includes time spent in layovers or consists solely of time in the air.
- Some flights require travelers to purchase paper tickets or some travelers may even prefer paper tickets. So if Dora wants paper tickets, she would need to determine how much the source (Orbitz, Expedia, etc) charges for paper tickets and add that to her price

- Dora is an American citizen and might require a visa for some destinations, or even transit visas for some stopovers. Thus, she would need to factor in potential visa fees to get a more complete idea of how much her trip is costing.

1.3 – The problem of heterogeneous data source integration and the ECOIN approach

The context differences that Dora faces is an example of the real-world problem that arises when integrating data from disparate, heterogeneous data sources. In the past, heterogeneous data source integration meant integrating databases that had context differences. But within the last decade, the problem has not only become more common but has become more varied as the types of data sources have proliferated.

The COntext INterchange (COIN) group at MIT addresses this problem with an approach known as context mediation [9]. This section presents a high level description of the COIN approach. For a detailed discussion of how context knowledge and data semantics are represented and reasoned about in the ECOIN system, we refer the reader to [2, 9, 10].

To address heterogeneous data source integration, the ECOIN system supports the implementation of *applications*: groups of heterogeneous data sources tied together such that they can be queried without worrying about context (i.e. without worrying about how to correctly interpret data from the different sources). ECOIN does context mediation that returns the data from the different sources to the user in his own context.

ECOIN supports the notion of context and context mediation through the use of *ontologies*, which contain *semantic types*, *attributes*, *modifiers* and *modifier values*. We explain these terms below either through direct definitions or by example.

An *ontology* (sometimes termed object model, application domain model, domain diagram, etc) is an explicit description of how to conceptualize the objects of some domain and the relationships between those objects [11]. For example, consider the domain of an application that determines airfares for travelers given the dates and destinations of travel. If we were to conceptualize such a domain, we would think of objects such as trip, airline, city, etc. Each of these objects that represents some concept is called a *semanticType*.

The relationships between these objects are captured by *attributes* and *modifiers*. For example, the semantic type, `trip`, would have attributes such as `airline`, `destination`, etc. The bottom of figure 1.3 below shows a small portion of the airfare ontology. We see the semantic types `trip`, `airline`, `city`, `moneyAmount`. The labeled arrows connecting these semantic types indicate the attribute relationships.

Modifiers tell ECOIN how to interpret a *semanticType*. Namely, the *modifiers* of a *semanticType* can take on one of many *modifier values* and it is the *modifier value* that tells ECOIN what the *semanticType* means. For example, in the Airfare example, the

semanticType, *moneyAmount*, would have a *modifier*, *currency*, that has a *modifier value* of “USD” for a US *context* and a *modifier value* of “GBP” for a British *context*.

A *context* is the set of assumptions regarding the interpretation or meaning of data. Users as well as sources can have their own contexts. Thus for each *modifier*, there exist *conversion functions* to convert data objects from one *context* to another.

Next, we have the data sources. ECOIN models all data sources as relations in a database regardless of whether the source actually is a database table. This is made possible by technologies such as Cameleon [7] (also developed by COIN) that allow semi-structured and even unstructured web pages to be seen as relational databases. For example, in figure 1.3 below, data from the Orbitz airfare aggregator website is extracted by Cameleon and modeled as if it were coming from a database table called `orbitz` with columns such as `airline`, `destination`, `price`, etc.

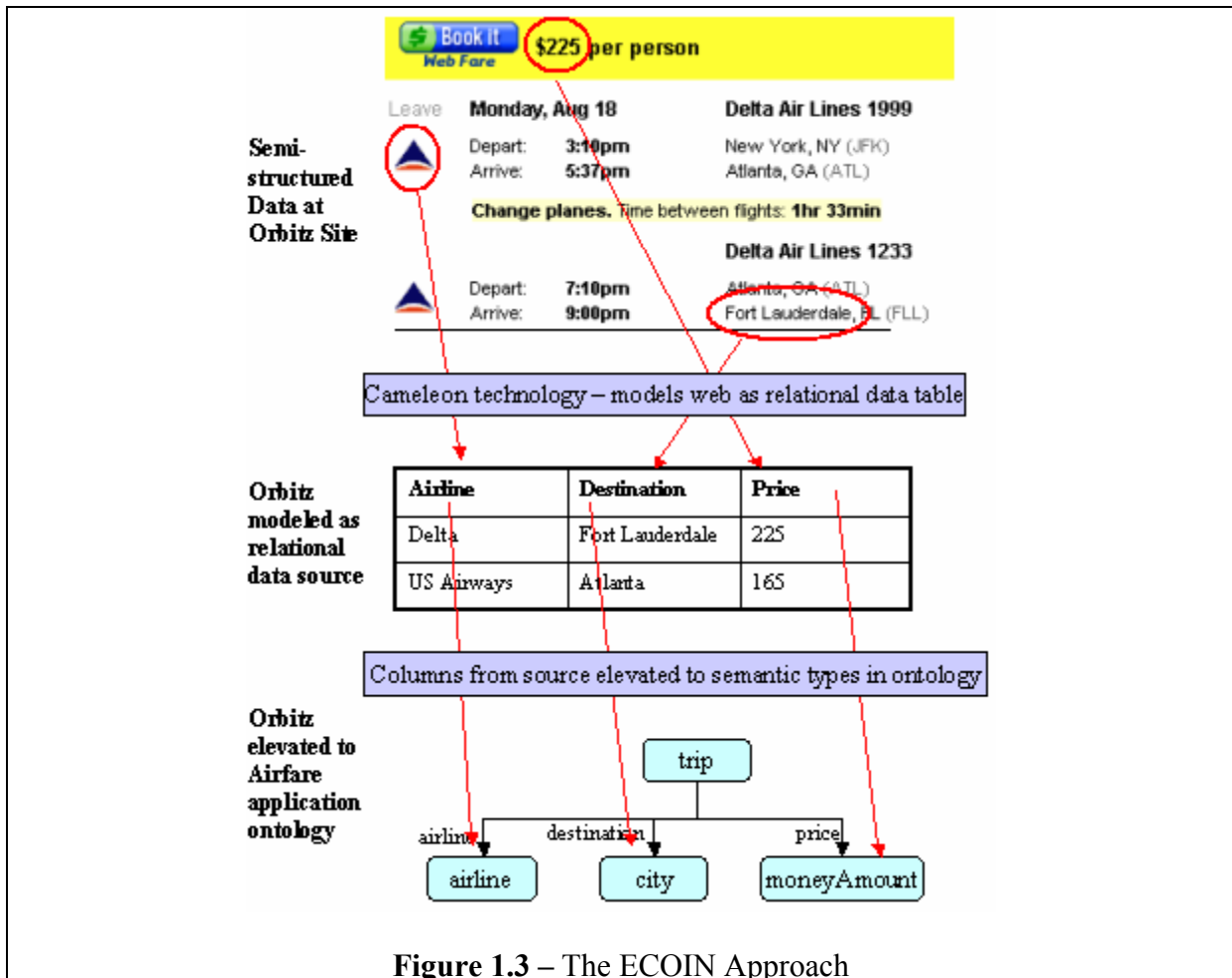


Figure 1.3 – The ECOIN Approach

ECOIN *elevates*, or maps, the columns from the data sources to *semanticTypes* in the ontology. For example, in figure 1.3, we see that the destination column from Orbitz is elevated to the *semanticType*, *city*.

To summarize, the semantic types, attributes and modifiers constitute an ontology that conceptualizes some domain. The modifier values and the conversion functions constitute the context information. The data sources and the elevation information that maps the columns of a data source into the ontology constitutes the source information. Together, the ontology, context and source information constitute an ECOIN application that supports context mediation across the data sources of that application.

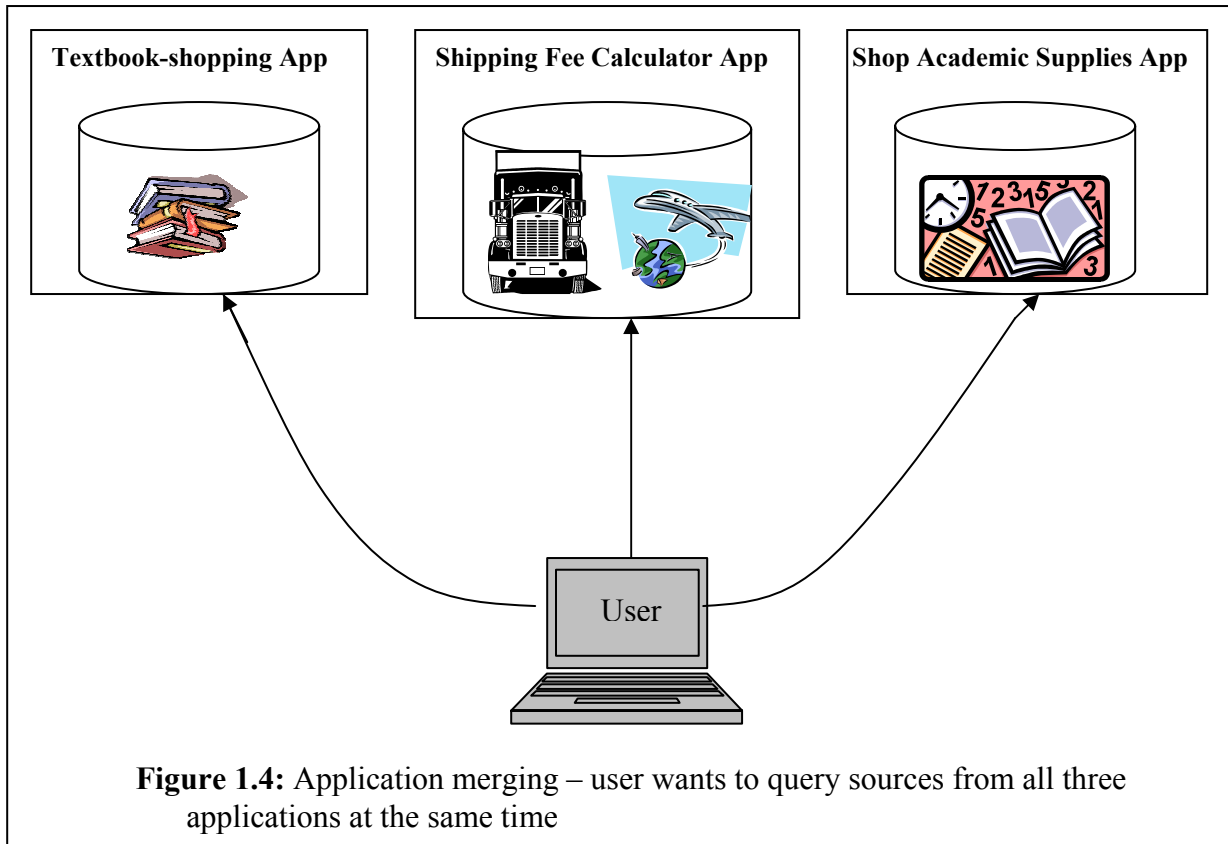
ECOIN stores all this application information as Prolog¹ rules in a plain text *application prolog file* (referred to as, “application file”). The component that makes use of all this information from the application file is called the abduction engine.² It takes in queries from the user and does context mediation to form a context-sensitive query to be sent to the data sources. After the query is sent to the data sources, the results are returned to the user in his own context (any conversions that are needed to transform data from the source contexts to the users’ context are automatically done).

1.4 - The Application Merging Problem

As discussed, an ECOIN application elevates a set of data sources to a common ontology, such that the data sources can be queried in a context-blind manner. For example, an ECOIN application for online textbook shopping may contain two data sources, i.e. two vendors. The user can query price and availability information from these two sources and have the price and availability results returned to him in a context that he understands. But what if the user learns of an application that allows general academic-supplies shopping, or an application that provides text-book data from two new vendors, or an application that calculates Fed-Ex or UPS shipping fees for a product. These three applications would be very interesting to the user because they either complement his original textbook-shopping application very well or expand the original application space in a useful manner. For example, while shopping for textbooks, a user might want to buy a book bag. Then he might want to have it all shipped to him. Rather than having to deal with different applications to query all the sources for his needs, it would be better to have it in one place. Given such scenarios, we are faced with the interesting problem of **ECOIN application-merging**.

¹ a rule-based logic programming language

² The abduction engine is the module (also written in Prolog) that contains all the logic for taking in a query that does not account for context and outputting a query sensitive to context (i.e. contains all the necessary context conversion functions). The abduction engine’s input is the “context-blind” query and as it runs it refers to the application’s prolog file, a constraints file, a couple of helper files (for example, a file containing dijkstra’s algorithm). The output is then a context-sensitive query. For a formal, detailed discussion of abduction and the abduction engine, see [aykut’s thesis].



1.4.1 - Virtual Merging versus Materialized Merging

What do we mean by application-merging? Is it the same as ontology-merging in general (i.e. merging the ontologies of the two applications)? Not exactly. Application merging proposes a “virtual” approach to ontology merging called **context-linking** while the traditional approach to ontology merging can be termed as “materialized” ontology merging.

Materialized ontology merging is done if the intention is to discard the original ontologies and use only the merged ontology in the future (a completely new ontology “materializes”).

Virtual ontology merging is done if the intention is to have the ontologies of the two applications *persist unchanged*. The new merged application gives the “virtual” appearance of one ontology and allows sources from both applications to be considered part of one new application. We discuss virtual versus materialized merging in much greater detail in Section 3.1.

1.4.2 - Motivation

This thesis’ research into application merging is driven by several motivations:

1. Value to be gained by merging applications

An application may find it very beneficial to have access to the set of sources that are part of another application. Or an application may be able to leverage the capabilities of another ontology by being merged with it. For example, consider the US textbook shopping application mentioned earlier and a Global Aggregator application that shops for a specific product in various countries, taking into account currencies and import/export taxes. The merged application would inherit the currency and import/export tax awareness and would stand ready to have international textbook sources added to it.

2. Re-use

With numerous applications and ontologies being developed independently, there is a lot of benefit to be derived from re-using work that has already been done. Merged ECOIN applications seek to re-use portions of the ontologies, contexts and application code of existing ECOIN applications.

3. Better fit with how ontologies/applications are developed in real life

In the real world, designers of applications (and ontologies) rarely have a broad enough vision to predict what will be desired in the future - ontologies, applications, and standards are constantly being developed and evolved by countless independent parties. It is better to adapt to this reality and design small, relevant applications and merge them with other applications as needed than to try to predict and pre-plan large, comprehensive applications.

4. Easier problem to solve than actually merging ontologies

The general problem of ontology merging is a difficult artificial intelligence problem. Having to determine relationships between entities of two arbitrary ontologies requires heuristics that approximate semantic understanding of the entities, which is difficult. The context linking approach thus bites off a smaller chunk of the problem: its goal is only to get two ECOIN applications to work together. Thus it reconciles parts of the two ontologies only when they would cause incorrect behavior or if there is some capability to be gained – otherwise it tolerates redundancy and even irrelevance in the new “virtually merged” ontology.

1.5 - Objectives and Roadmap

The main objectives of this thesis are three:

1. Present and discuss the Context Linking approach to application merging
2. Study and evaluate that approach by merging independent, realistic applications
3. Design a tool that facilitates the context linking/merger axiom creation process.

The rest of this thesis is organized as follows. Chapter 2 presents the applications that will be the subject of the merging case study. Chapter 3 discusses Ontology and Application merging in detail, presenting context linking and merger axioms in meticulous depth. Chapter 4 proposes a tool to facilitate merger axiom creation and presents a design for it. Chapter 5 presents possible extensions and improvements to the tool. Chapter 6 discusses related research efforts by other groups and chapter 7 presents conclusions and looks to the future.

2. The Applications: Airfare, Car Rental and Their Merger

In Chapter 1, we presented motivation for merging multiple applications. In the rest of this thesis, we shall be discussing merging in detail and much of it will be through a case study of merging two real ECOIN applications: Airfare and Car Rental. Accordingly, this chapter presents the details of these two applications.

2.1 The Airfare Application

Chapter 1 discusses Dora's motivation for creating an Airfare application. In summary, having to deal with multiple sources means dealing with context differences between the sources. The Airfare Application allows users to browse various sources and see prices in their own context. Chapter 1 also discusses what the context issues are and below we restate (in brief) only those context issues that the application will resolve:

- What does price include? Just airfare? Taxes? Service fees?
- What currency is the price in?
- If paper tickets are wanted, how much of a charge should be added to price?
- Does the understanding of price include a visa fee (if the destination country charges such a fee)?

2.1.1 – Steps to development

The steps Dora takes in creating the airfare application are generally as follows:

- 1) Determine the various sources the application will need to draw upon (Orbitz, Expedia, other travel sites, currency converter website, etc)
- 2) Peruse the sources to determine all the context differences and to determine what additional sources might be needed to help resolve those differences
- 3) Determine what data will be needed from the sources, what data will required from the user, and think of these as columns in data tables that represent each of the sources. “Wrap” the web sources with cameleon spec files (cite paper) to enable data extraction from the web sources.
- 4) Develop an ontology (domain model) that conceptually describes the airfare application space and elevate the data columns of the sources to semanticTypes (entities) in the ontology.
- 5) Create a set contexts, each being a particular perspective of the application data. These perspectives are held either by the sources (source contexts) or by users (receiver contexts).
- 6) Determine conversions between the various contexts i.e. for each modifier, develop functions that convert from one modifier value to another.
- 7) Formalize all of the above in appropriate ECOIN format – namely, the application file capturing ontology, source, context and conversion information and the schema file (written in XML) capturing the schema of all the sources. (see Appendix A for Prolog and files capturing all this information)

2.1.2 - Sources

The following is a summary of the sources in the Airfare application:

Category of Source	Source Name
Airfare Sources (all from the Web and wrapped with Cameleon)	<ul style="list-style-type: none"> - Expedia - http://www.expedia.com - Orbitz - http://www.orbitz.com - Itn (owned by American Express) - http://www.itn.net - Yahoo - http://travel.yahoo.com - TravelSelect (a UK source) - http://www.travelselect.com
Other Information sources	<ul style="list-style-type: none"> - Olsen currency converter - http://www.oanda.com/convert
Oracle tables created for utility	<ul style="list-style-type: none"> - ServiceFees (service fees charged by various airfare aggregators) - PaperFees - (paper ticket charges of various airfare aggregators) - VisaFees – (info from various embassy websites regarding visa fees to various countries, given various citizenships)

Table 2.1 – Sources of Airfare Application

2.1.3 - Ontology and Elevations

The ontology for the Airfare application is as follows:

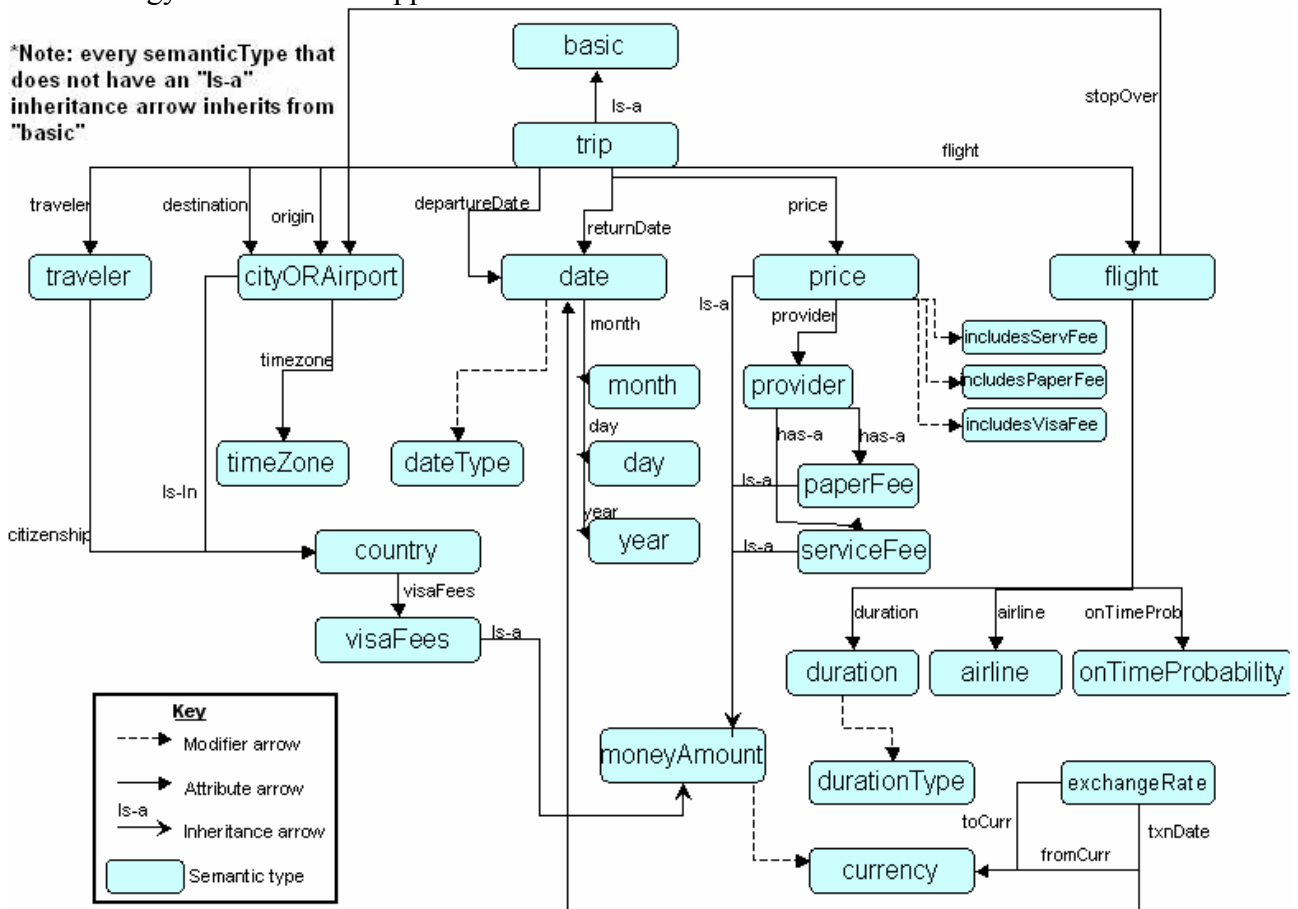


Figure 2.1 – Airfare Ontology

The semantic type, *basic*, is the starting point of the ontology and all inheritance chains eventually lead back to it. The ontology details all the entities (semantic types) in the application space and models the relationships between them. The various “columns” of data

from all the sources are elevated to semantic types in the ontology. A following is a summary of these elevations:

Source	Elevations (Column in data source → semantic type)
Elevations from Airfare sources	<ul style="list-style-type: none"> - 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	<ul style="list-style-type: none"> - Exchanged (i.e. fromCurrency) → currency - Expressed (i.e. toCurrency) → currency - Rate → exchangeRate - Date (i.e. txnDate) → date
Elevations from service fees table	<ul style="list-style-type: none"> - Provider (i.e. site that found fare, ex. Orbitz) → provider - Service Fee → serviceFee
Elevations from paper_fees table	<ul style="list-style-type: none"> - Provider (i.e. site that found fare, ex. Orbitz) → provider - Paper Fee → paperFee
Elevations from visa_fees table	<ul style="list-style-type: none"> - Citizenship (of traveler) → country - Destination (of traveler) → country - Visa Fee → visaFee

Table 2.2 – Airfare Application Elevations

2.1.4 - Modifiers and Context

The most interesting part of the ontology are the semantic types that have modifiers because it is the modifiers that allow the existence of multiple contexts. Indeed, the existence of modifiers is what distinguishes ECOIN ontologies from ontologies in the traditional sense – ECOIN ontologies contain modifiers while traditional ontologies are limited to semantic types and attributes (or equivalents thereof). In this ontology, `moneyAmount`, `price`, `duration` and `date` have modifiers.

`MoneyAmount` has the modifier, `currency`. All `semanticTypes` that inherit from `moneyAmount` (i.e. `price`, `serviceFee`, `paperFee`, `visaFee`) also inherit `currency`.

price has the modifiers, includesServFee, includesPaperFee and includesVisaFee all of which can take a value of either “yes” or “no” based on whether the understanding of price is to include the fee in question.

Duration has the modifier durationType and date has the modifier dateType. These were created to support multiple understandings of duration and date - namely, a duration that includes stopover time versus only flight time and a US date type versus a European date type. However, in this version of the application, these modifiers were not implemented in the application prolog file.

The application defines several contexts – one for each source and two user contexts. Each context is defined by the set of modifier values that captures the assumptions that particular source (or user) makes when interpreting the data. The following is a table that summarizes the modifier values for all the contexts:

<u>Context Type</u>	<u>Context Name</u>	<u>includesServiceFee</u>	<u>includesPaperTktCharge</u>	<u>includesVisaFee</u>	<u>Currency</u>
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

Table 2.3 – Airfare Application Context Table

2.1.5 - Sample Queries

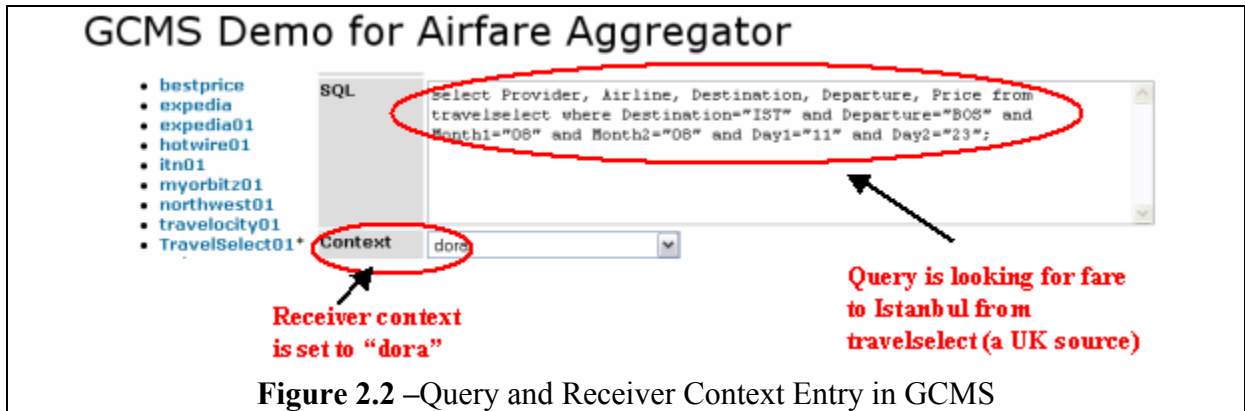
Going back to Dora’s planning of her trip to Istanbul, we walk through one query she might run during her search:

Select Provider, Airline, Destination, Departure, Price from travelselect where Destination="IST" and Departure="BOS" and Month1="08" and Month2="08" and Day1="11" and Day2="23";

Below is an illustration of the steps her query goes through on GCMS:

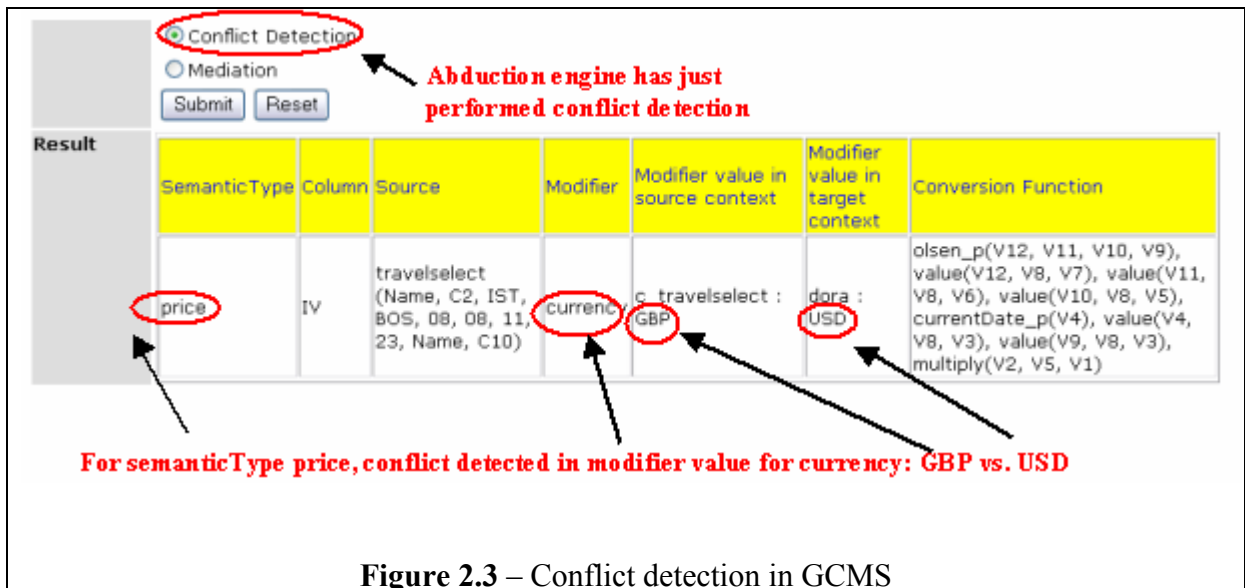
1. Query and Receiver Context Entry

GCMS sends a user's query to the **ECOIN abduction engine** along with the user's context. Next, the abduction engine sees what source the user wants to query and determines the source context.



2. Conflict Detection

Using ontology and elevation information drawn from the application file, the abduction engine determines the semantic types of the data that are desired from the source. Looking at the user's query and then at the elevations table (Table 2.2), we see that the semantic types in this case are provider, airline, cityOrAirport and price. The engine checks if those semantic types have modifiers associated with them and if so, it determines if there are any differences in the modifiers' values in the source context versus the receiver context (i.e. are there any context differences?). If we look at the ontology diagram again we see that price has modifiers. Next, if we look at the context table (Table 2.3) we see that there is a conflict in the values of currency, includesServFee and includesPaperFee between the Travelselect context and Dora context.



3. Mediation and Revised Query

If there are conflicts, the abduction engine revises the user's query to contain conversion functions that will reconcile the differences between the user's context and the source's context. Note how different (and longer!) the query below is compared to the original user's query in Step 1. Without context mediation, not only would the user have to write long, tedious queries, but would need to be aware of the different contexts the sources assume and would need to know or lookup how to convert between contexts (for example, how much of a service fee to add).

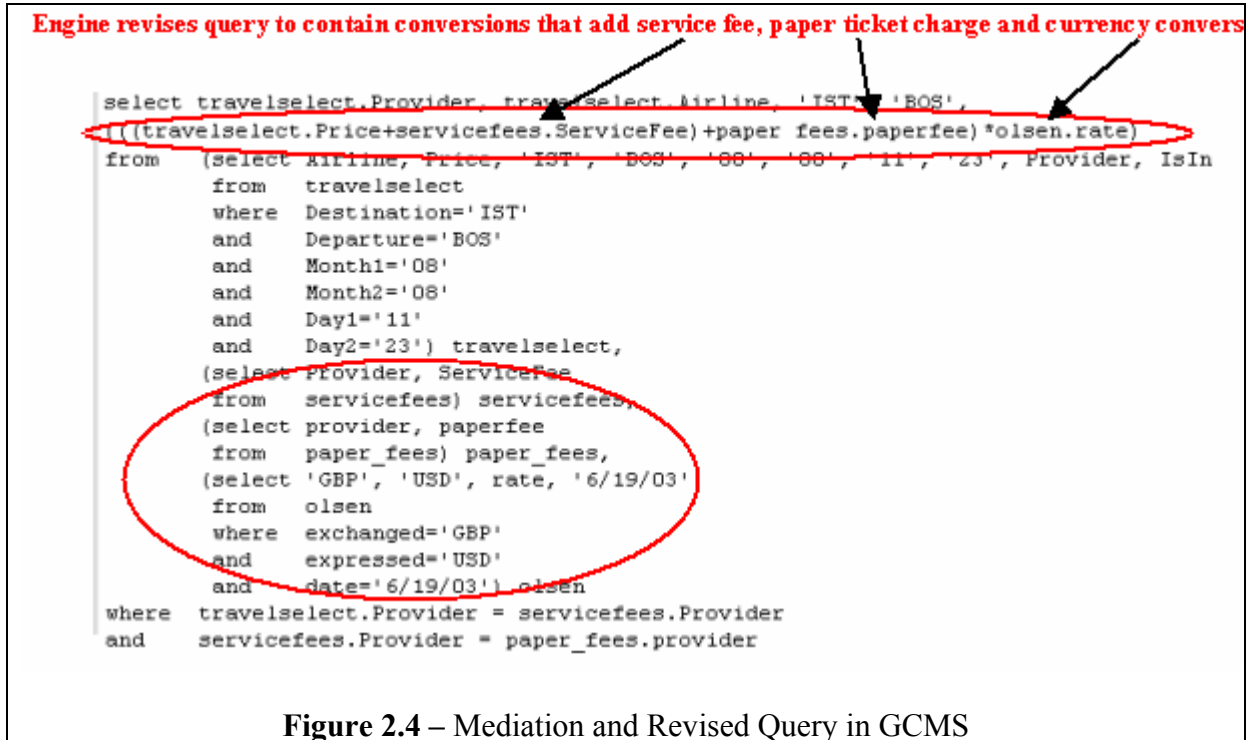


Figure 2.4 – Mediation and Revised Query in GCMS

If Dora wanted to find the cheapest fares across all sources, she would write the following query:

```
select Provider, Airline, Destination, Departure, Price from yahoo where Destination="NRT" and Departure="BOS" and
Month1="Aug" and Month2="Aug" and Day1="23" and Day2="30"
UNION
select Provider, Airline, Destination, Departure, Price from expedia2 where Destination="NRT" and Departure="BOS" and
Date1="08/23/03" and Date2="08/30/03"
UNION
select Provider, Airline, Destination, Departure, Price from myorbitz where Destination="NRT" and Departure="BOS" and
Month1="Aug" and Month2="Aug" and Day1="23" and Day2="30"
UNION
select Provider, Airline, Destination, Departure, Price from traveselect where Destination="NRT" and Departure="BOS" and
Month1="08" and Month2="08" and Day1="23" and Day2="30"
UNION
select Provider, Airline, Destination, Departure, Price from itn where Destination="NRT" and Departure="BOS" and
Month1="Aug" and Month2="Aug" and Day1="23" and Day2="30";
```

There are five sources here and Dora can be blissfully ignorant of all the differences between each of the sources' contexts and her own. To get an inkling of how much work she's saving, we can see how many conversions, from adding fees to changing currencies, the generated context-sensitive SQL query is doing below:

```

select yahoo.Provider, yahoo.Airline, 'NRT', 'BOS', (yahoo.Price+paper_fees.paperfee)
from (select Airline, Price, 'NRT', 'BOS', 'Aug', 'Aug', '23', '30', Provider, IsIn
      from yahoo
      where Destination='NRT'
            and Departure='BOS'
            and Month1='Aug'
            and Month2='Aug'
            and Day1='23'
            and Day2='30') yahoo,
(select provider, paperfee
 from paper_fees) paper_fees
where yahoo.Provider = paper_fees.provider
union
select expedia2.Provider, expedia2.Airline, 'NRT', 'BOS', expedia2.Price+paper_fees2.paperfee)
from (select Airline, Price, 'NRT', 'BOS', '08/23/03', '08/30/03', Provider, IsIn
      from expedia2
      where Destination='NRT'
            and Departure='BOS'
            and Date1='08/23/03'
            and Date2='08/30/03') expedia2,
(select provider, paperfee
 from paper_fees) paper_fees2
where expedia2.Provider = paper_fees2.provider
union
select myorbitz.Provider, myorbitz.Airline, 'NRT', 'BOS', ((myorbitz.Price+servicefees.ServiceFee)+paper_fees3.paperfee)
from (select Airline, Price, 'NRT', 'BOS', 'Aug', 'Aug', '23', '30', Provider, IsIn
      from myorbitz
      where Destination='NRT'
            and Departure='BOS'
            and Month1='Aug'
            and Month2='Aug'
            and Day1='23'
            and Day2='30') myorbitz,
(select Provider, ServiceFee
 from servicefees) servicefees,
(select provider, paperfee
 from paper_fees) paper_fees3
where myorbitz.Provider = servicefees.Provider
and servicefees.Provider = paper_fees3.provider
union
select traveselect.Provider, traveselect.Airline, 'NRT', 'BOS',
(((traveselect.Price+servicefees2.ServiceFee)+paper_fees4.paperfee)*olsen.rate)
from (select Airline, Price, 'NRT', 'BOS', '08', '08', '23', '30', Provider, IsIn
      from traveselect
      where Destination='NRT'
            and Departure='BOS'
            and Month1='08'
            and Month2='08'
            and Day1='23'
            and Day2='30') traveselect,
(select Provider, ServiceFee
 from servicefees) servicefees2,
(select provider, paperfee
 from paper_fees) paper_fees4,
(select 'GBP', 'USD', rate, '7/10/03'
 from olsen
 where exchanged='GBP'
 and expressed='USD'
 and date='7/10/03') olsen
where traveselect.Provider = servicefees2.Provider
and servicefees2.Provider = paper_fees4.provider

```

```

union
select itn.Provider, itn.Airline, 'NRT', 'BOS', ((itn.Price+servicefees3.ServiceFee)+paper_fees5.paperfee)
from (select Airline, Price, 'NRT', 'BOS', 'Aug', 'Aug', '23', '30', Provider
      from itn
      where Destination='NRT'
            and Departure='BOS'
            and Month1='Aug'
            and Month2='Aug'
            and Day1='23'
            and Day2='30') itn,
(select Provider, ServiceFee
 from servicefees) servicefees3,
(select provider, paperfee
 from paper_fees) paper_fees5
where itn.Provider = servicefees3.Provider
and servicefees3.Provider = paper_fees5.provider

```

The final result returned is as follows:

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	 American Airlines®	NRT	BOS	4602.7
Expedia	American	NRT	BOS	1045

Figure 2.5 – Result of query to all sources for cheapest airfare

These results are all in Dora’s context. That is, the prices quoted are all in US dollars, include the service fee, the paper ticket charge and a visa fee to go to Japan regardless of whether Orbitz, Yahoo etc shared such an understanding of price.

2.1.6 - Future expansion: accommodated by the ontology or left up to merging?

If we look at the ontology diagram in Figure 2.2 and then look at the elevations in Table 2.2, we note that there are semantic types in the ontology that are not mentioned in the elevations table. This means there are semantic types and attributes that the ontology provides but are not currently being used in the application. This is because ontologies are purposely created to be broad and general – an idealistic rendering of the application space where some of the features may not necessarily exist but will perhaps be added in the future.

In this case, we have `duration`, `durationType`, `dateType`, `onTimeProbability` and `timeZone`. `duration` and `dateType` were discussed earlier.

`OnTimeProbabilities` and `timeZone` can be used when trying to figure out the chances of missing a connection. To implement both of these features would require gleaning stopover information, timezone information and flight timeliness information from the sources. All of this information is usually available and would make for a very interesting application expansion.

Overall, however, we note that the ontology is not that much broader in scope than the actual application implementation – there is not that much left for future expansion. This is intentional because, as a ECOIN application, in the future, Airfare can simply be merged with other applications that provide interesting expansion or features. This is a much simpler way of creating ontologies because it saves the developer from having to predict what directions the application will need to go in the future and allows him to focus on the current needs of the application.

2.2 Car Rental Application

In this section, we describe the second ECOIN application in this case study of merging – Car Rental. The sections below present the motivation, contexts, ontology, sources and a sample run of the application. Appendix B contains the application Prolog file that captures all of the ontology, source and context information of Car Rental.

2.2.1 – Motivation and Context Differences

Joe is another person in the group who travels a lot. However, he is not as adventurous as Dora and prefers to stay within the country. Rather than fly, he usually rents cars and drives everywhere and thus is an avid user of car rental websites. At these sites, he enters the date, place and time he would like to pick up and return the car and the site quotes him various rates from different rental companies. Once again there are several context issues that Joe faces in searching the different sites. A summary of the issues he faces is:

- What is the time period that the rate is based on? Suppose Joe wants to rent for six days. One site may quote him a rate of \$24 (per day) while another may quote him \$150 (per week) because some companies give a weekly rate even when renting for as little as five days (as long as a weekend falls among those five days). To compare them meaningfully, Joe must convert those rates and determine what his total cost would be. The mental arithmetic can be difficult, bothersome and error-prone when multiple sites are queried.
- What about all the fees, taxes, insurance? Here is where the real nightmare is – different states have different laws about taxes and insurance while different rental companies have a multitude of fees they may charge. Clicking through multiple screens of various rental websites to determine the various taxes and fees is no quick task.
- Renting from a location in the city or an airport? Joe knows that the biggest selection of cars in a city is usually at the airport and so he often has to look up the airport code of a city in order to rent from there. It would be very convenient if he could simply enter the city and state that he would like to rent from and have that automatically be converted to an airport code.

- Some sites accept dates in numeric format while others expect otherwise (i.e. 08 vs. Aug)

Overall, there are several issues that Joe would have to deal with every time he does a car rental search online. Thus, inspired by his friend, he too decides to create a “context interchange” application that can potentially alleviate most, if not all, of the context issues for him.

2.2.2 – Sources

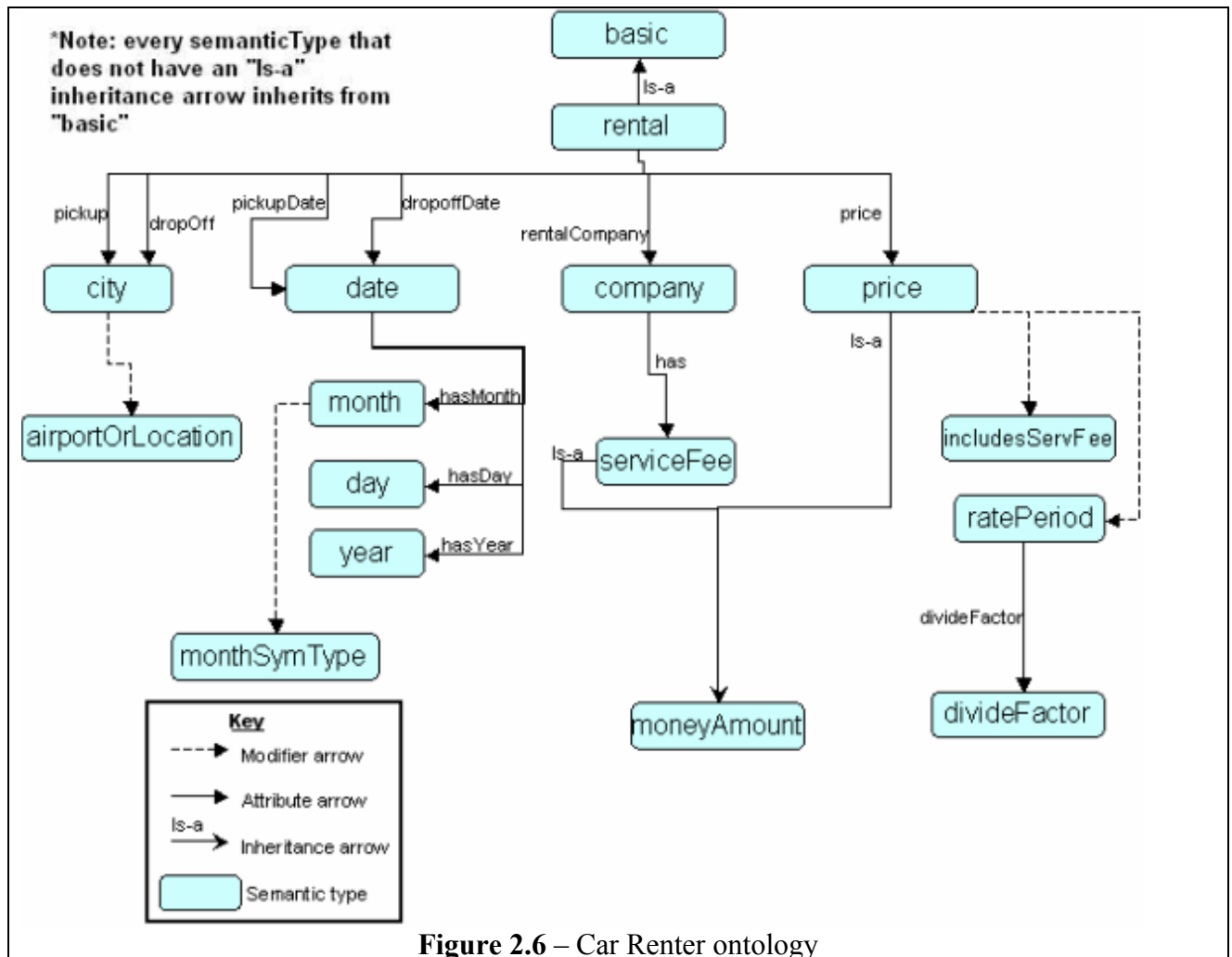
The application Joe creates is called Car Rental. The following is a summary of the sources in Car Rental:

Category of Source	Source Name
Car Rental Sources (all from the Web and wrapped with Cameleon)	- Expediacar – http://www.expedia.com - Yahooocar - http://travel.yahoo.com - Qixocar – http://www.usahotelguide.com/nexres/cars/
Other Information sources	- http://www.logisticsworld.com/airports.asp (cameleon)
Oracle tables created for utility	- month_symbol_converter (converts from numeric to three-letter symbol) - rate_period_dividefactors (divide factors to convert from one rateperiod to another)

Table 2.4 – Car Renter Sources

2.2.3 - Ontology and Elevations

The ontology for the Car Rental application is as follows:



The various “columns” of data from all the data sources are elevated to semantic types in the ontology. The following is a summary of the elevations:

Source	Elevations
Elevations from Airfare sources	<ul style="list-style-type: none"> - 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	<ul style="list-style-type: none"> - Location → city - AirportCode → city
Elevations from month_sym_converter	<ul style="list-style-type: none"> - Mm_number → month - symbol → month
Elevations from rateperiod_dividefactors	<ul style="list-style-type: none"> - Rateperiod (ex: ‘weekly’, ‘monthly’, etc) → ratePeriod - DivideFactor → divideFactor

Table 2.5 – Car Renter Elevations

2.2.4 - Modifiers and Context

There are three semantic types in the Car Rental ontology that have modifiers: `month`, `price` and `city`.

`month` has the modifier, `monthSymType` which can take on the values “numeric” or “three-letter” depending on whether the context in question represents months with a two-digit number or a three-letter symbol.

`Price` has the modifiers `includeServFee` and `ratePeriod`. `includeServFee` can take on the values “yes” or “no” depending on whether a service fee is included or not. `ratePeriod` is a dynamic modifier. For each rate that is quoted, the `ratePeriod` might differ because the rates different companies quote might be based on different rate periods. Thus for each result returned, the modifier value could be different – the abduction engine must lookup what `ratePeriod` (‘monthly’, ‘weekly’, ‘daily,’ etc) was returned and use that as the modifier value. So if a user wants to rent from August 5 to August 14, one result that might be returned is a rental rate of \$270 with a “weekly” rate period. Now suppose the user wants his `ratePeriod` to be “total” i.e. the rate for the total nine days. In that case he would have to divide the \$270 by seven to get the rate per day and then multiply by nine days to get the price for the entire period.³

³ This discussion makes a few simplifying assumptions. For example, one thing we assume is that the rental company would be willing to give the two days beyond the first seven days at the weekly rate. We are making simplifying assumptions because the purpose here is not to create a car rental application that captures every existing subtlety and can immediately be used in the real world. Rather, the goal is to give a flavor of the context

city has the modifier `airportOrLocation` which can take the value of “airport” or “location” depending on whether the source/user refers to locations in “city, state” format or with an airport code.

Using these modifier values, the application defines several contexts – one for each source and two user contexts. The following is a table that summarizes the modifier values for all the contexts:

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"

Table 2.6 – Car Renter Context Table

2.2.5 - Sample Query

To get an idea of the conversions involved, we look at a query Joe might run to find the cheapest rental rates across multiple rental sources:

```
select Price, Company, Rateperiod from expediacar where Pickup="San Francisco CA" and Dropoff="same" and Date1="08/10/03"
and Date2="08/13/03" and Month1="Aug" and Month2="Aug" and Day1="07" and Day2="10"
UNION
select Price, Company, Rateperiod from qixocar where Pickup="San Francisco CA" and Dropoff="same" and Month1="Aug" and
Month2="Aug" and Day1="07" and Day2="10"
UNION
select Price, Company, Rateperiod from yahoocar where Pickup="San Francisco CA" and Dropoff="same" and Month1="Aug" and
Month2="Aug" and Day1="07" and Day2="10";
```

The query the abduction engine returns is as follows. Note all the automatic airport code conversions, month symbol conversions and service fee additions:

```
select expediacar.Price, expediacar.Company, expediacar.Rateperiod
from (select 'San Francisco CA', airportcode
      from airport_code_lookup
      where location='San Francisco CA') airport_code_lookup,
(select 'same', airportcode
      from airport_code_lookup
      where location='same') airport_code_lookup2,
(select mm_number, 'Aug'
      from month_symbol_converter
      where symbol='Aug') month_symbol_converter,
(select mm_number, 'Aug'
      from month_symbol_converter
```

issues that exist in the car rental domain in order to demonstrate context mediation. If we wanted to solve this problem thoroughly, we would capture the sundry subtleties of rateperiods more carefully.

```

where symbol='Aug') month_symbol_converter2,
(select Pickup, Dropoff, '08/10/03', '08/13/03', Month1, Month2, '07', '10', Price, Company, Rateperiod
from expediacar
where Date1='08/10/03'
and Date2='08/13/03'
and Day1='07'
and Day2='10') expediacar
where month_symbol_converter.mm_number = expediacar.Month1
and month_symbol_converter2.mm_number = expediacar.Month2
and airport_code_lookup.airportcode = expediacar.Pickup
and airport_code_lookup2.airportcode = expediacar.Dropoff
union
select (qixocar.Price+9.99), qixocar.Company, qixocar.Rateperiod
from (select 'San Francisco CA', airportcode
from airport_code_lookup
where location='San Francisco CA') airport_code_lookup3,
(select 'same', airportcode
from airport_code_lookup
where location='same') airport_code_lookup4,
(select mm_number, 'Aug'
from month_symbol_converter
where symbol='Aug') month_symbol_converter3,
(select mm_number, 'Aug'
from month_symbol_converter
where symbol='Aug') month_symbol_converter4,
(select Pickup, Dropoff, Month1, Month2, '07', '10', Price, Company, Rateperiod
from qixocar
where Day1='07'
and Day2='10') qixocar
where month_symbol_converter4.mm_number = qixocar.Month2
and airport_code_lookup3.airportcode = qixocar.Pickup
and airport_code_lookup4.airportcode = qixocar.Dropoff
and month_symbol_converter3.mm_number = qixocar.Month1
union
select yahooocar.Price, yahooocar.Company, yahooocar.Rateperiod
from (select 'San Francisco CA', airportcode
from airport_code_lookup
where location='San Francisco CA') airport_code_lookup5,
(select 'same', airportcode
from airport_code_lookup
where location='same') airport_code_lookup6,
(select Pickup, Dropoff, 'Aug', 'Aug', '07', '10', Price, Company, Rateperiod
from yahooocar
where Month1='Aug'
and Month2='Aug'
and Day1='07'
and Day2='10') yahooocar
where airport_code_lookup5.airportcode = yahooocar.Pickup
and airport_code_lookup6.airportcode = yahooocar.Dropoff

```

The final result returned is as follows:

Price	Company	Rateperiod
21.99	Dollar Rent A Car	Day
21.99	Expedia.com	Day
22.99	Hertz	Day
22.99	home	Day
23.89	PAYLESS	Daily
23.99	Avis	Day
23.99	flights	Day
25.89	Payless	DAILY
26.99	FOX RAC	Daily
27.99	Hertz	Day
27.99	cars	Day
27.99	hotels	Day
32.99	National Car Rental	DAILY
35.99	NATIONAL	Daily
37.95	ALAMO	Daily
39.49	Enterprise	DAILY
41.99	Hertz	Day
41.99	vacations	Day

Figure 2.7 – Results of Query in Car Renter to all sources

2.3 - Merging Airfare and Car Rental

The airfare and car rental applications form excellent complements of each other and the logical question arises: is it possible to have a general travel application that searches for both airfares and car rental rates? The answer is yes – ECOIN application merging allows large applications to be built from multiple small applications and allows reuse of relevant portions of existing ontologies.

But application merging is not just for building larger applications from smaller applications. There are three goals that can potentially be accomplished when merging any two applications. We present these goals below and also explain them in light of the Airfare and Car Rental Applications.

2.3.1 - Goals of Merging Two Applications

Goal 1:

Seamless source access: Sources from both applications will be available in one application. For example, a user would be able to use one, consolidated query to access sources from both the airfare and the car rental application to find the lowest airfare and cheapest car at his destination:

Give me the airline and price from expedia and a car rental company and rental rate from yahooCar for the cheapest airfare and car rental for a trip from Boston, MA to San Francisco, CA from Jun 13th 2003 to Jun 19th 2003.

Without merging, the sources of an ECOIN application are available only when running that specific application. If a user finds a set of sources in application Y that he believes would be relevant in application X, he can only access those sources by running application Y. If he wanted to access Application Y's sources from application X, he would have to add them to application X and elevate them to application X's ontology. Achieving Goal 1 would eliminate this limitation - Sources from both applications would be available in one application without the need to elevate those sources or modify either ontology.

Goal 2:

Cross-fertilization of contexts: Use the context capabilities of one application to benefit the other application. For example:

- Use currency conversion available from Airfare application to benefit car rental (which has no notion of currency) so merged application will support the addition of new, international car rental sources
- Use city name to airport code conversion available from Car Rental application to benefit Airfare application (which deals only in airport codes) so that the merged application allows airfare queries in which the user does not need to know airport codes.

There are several types of context differences that are general enough problems that we see them appear in many application domains. Currency conversion, sales tax calculation, European to American date format conversion are a few such examples. Beyond these, there are other, less general context issues that may be solved in one application and would be useful in another application. For example, calculating shipping costs based on weight and dimension might be solved in a textbook shopping application and would be useful if imported into some other shopping application. The ideal solution would avoid duplication of work and re-use contexts, sources and application code available in other applications to solve the needs of an application in question. This is what Goal 2 seeks - cross-fertilization of contexts allows a developer to breed for the best possible genes from two applications.

Goal 3:

Value Added Benefits: Extend the new merged application to add value beyond what the two applications can provide. For example adding a new source or a new context that makes sense in the merged environment but would not make sense in the individual applications:

Add a new context called FlyAndRent to the merged Airfare and Car Rental Application. In this context, define price as sum of airfare and car rental price. As a result, in a query, the user can simply ask for price and without having to query an airfare source *and* a car rental source (such as in goal #1 above), he is quoted a price that includes both.

The addition of FlyAndRent to the merged application would demonstrate that a merged application allows extension of the two previous applications to provide features that could not have been provided by simply extending one (or both) of the independent applications.

The three goals of merging discussed in this section will come up repeatedly in the rest of this thesis and for the sake of simplicity, will be referred to as “Goal 1,” “Goal 2” and “Goal 3.”

3. – Merging Applications

The previous chapter discussed the Airfare and Car Rental applications, the motivation for merging them and the goals of merging. This chapter delves into the details of merging. It discusses two possible types of merging, materialized versus virtual. Then it goes into the details of virtual merging using context linking and presents an algorithm of such a process. Finally it presents the merging details of the Airfare + Car Rental study.

3.1 – Materialized Merging versus Virtual Merging

Succinctly defined, ontology merging seeks to establish correspondences among source ontologies and to determine the set of overlapping concepts, concepts that are similar in meaning but have different names or structure. Ontology merging also seeks to establish relationships between the concepts that are unique to each of the sources [14].

Ontology merging literature discusses two flavors of merging. It terms them **merging** and **alignment**. It uses the term **merging** when the ultimate goal is to create a single coherent ontology that includes the information from all the source ontologies. It uses the term **alignment** when the source ontologies must be made consistent and coherent with one another but kept separately [14]. As mentioned in Section 1.4.1, we identify this distinction in merging approaches as materialized versus virtual. In the sections below, we present these two approaches in more detail and justify the use of virtual merging for the purpose of ECOIN applications.

3.1.1 – Materialized Merging

Materialized merging begins with two ontologies and ends with one coherent, complete, new ontology that contains no redundancies i.e. the resulting ontology will not contain multiple semantic types describing the same (or similar) concept. For example, if ontology A contains `Price` and ontology B contains `Cost` to represent the same concept, then only one of those entities should exist in the merger of A and B. In the end, the result that is sought is the “materialization” of a single, new ontology that covers the domains of the two individual ontologies in an elegant manner – i.e. it should seem, as much as possible, as if the ontology had been developed from scratch with the goal of covering the union of the two domains and the two constituent ontologies are meant to be scrapped.

To achieve “materialized” merging, one needs to go through every single semantic type and attribute, determine its meaning, determine if there is some equivalent, some subclass, some superclass or some similar semantic type in the other ontology. If so, the redundancies have to be removed and the correct relationships have to be established (sub or superclass, homonym, synonym, etc [1]). For example, let us take a look at simplified versions of the Airfare and Car Rental ontologies (see Figure 3.1 on next page) and merge them using the materialized approach:

Looking at both ontologies, we see many similar semantic types. `trip` from Airfare and `rental` from Car Rental have many similar attributes: `destination` and `origin` are analogous to `pickup` and `dropoff`, `departureDate` and `returnDate` are analogous to `pickupDate` and `dropoffDate`, `price` is analogous to `price`, and `airline` is analogous to `rentalCompany`. Thus the semantic types these attributes point to can be considered synonyms (i.e. `date` from both ontologies are synonyms, `price` from both ontologies are synonyms, `airline` and `company` are synonyms, `cityOrAirport` and `city` are synonyms). Beyond these, there are even more analogous semantic types: `moneyAmount`, `serviceFee`, and `includesServFee`.

Given the above overlap in semantic type and attribute relationships, the two ontologies can be merged by considering `trip` a superclass of `rental` since `trip` is a more general concept (i.e. renting a car can be considered as going on trip). Those attributes and semantic types that exist below `rental` but do not exist below `trip` can simply be added to the merged ontology in the appropriate places. The overall result appears at the bottom of Figure 3.1 below:

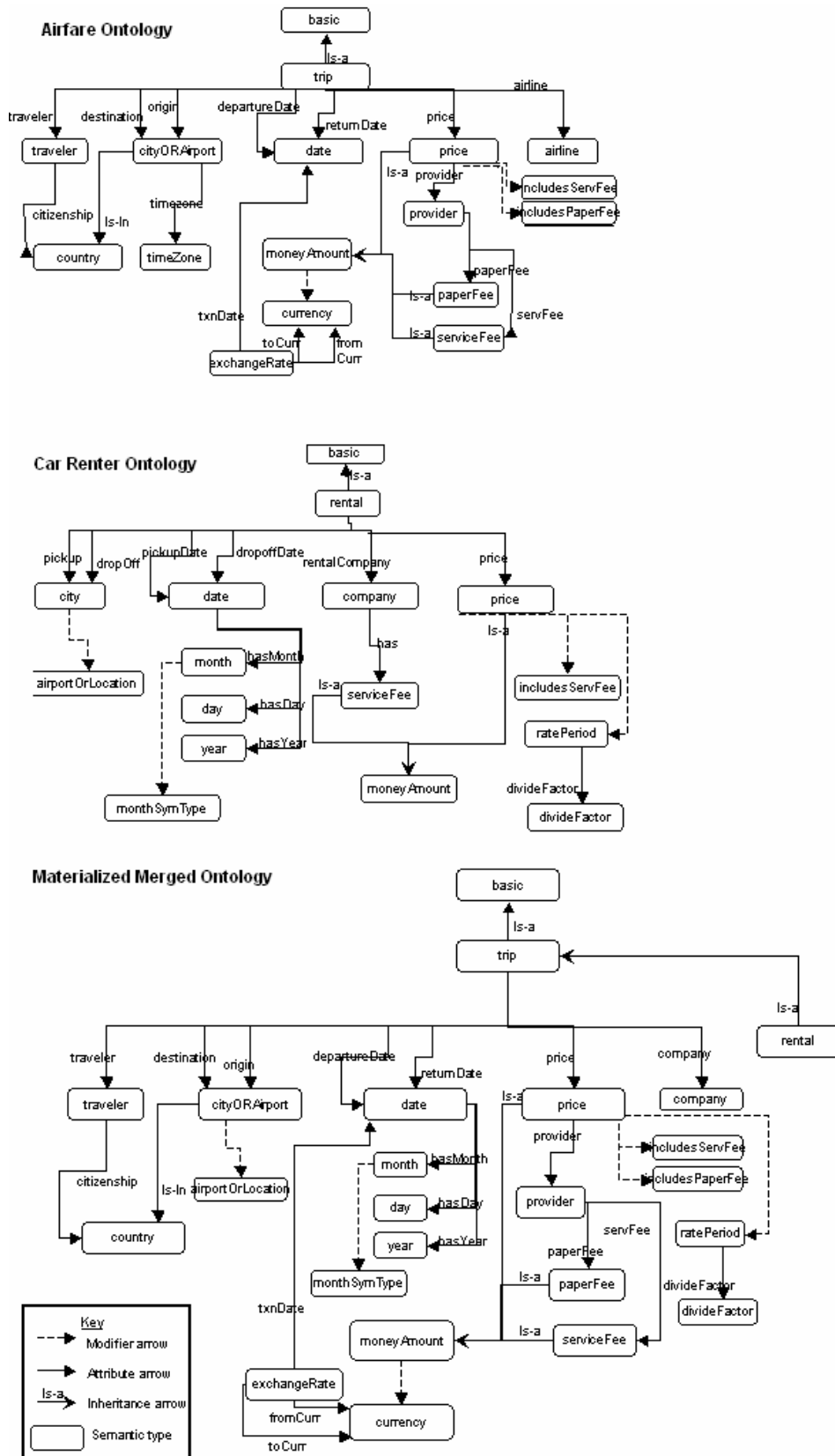


Figure 3.1 Airfare, Car Renter and Materialized Merged Ontologies

As discussed, `rental` now inherits from `trip`. Virtually none of the attribute names of `trip` were modified because they were analogous to the attribute names of `rental` even if the naming now seems a bit awkward when applied to `rental`. For example, `departureDate` works well for airline trips but is somewhat of an awkward name for the pickup date of a car rental. However, it is not too much of a stretch and is acceptable because one can conceive of the “departure date” of a car rental trip. The only attribute name (and semantic type) that was modified was `airline`, which was changed to the more general, `company` because `company` can represent the airline company as well as the car rental company. Looking at the rest of the ontology, we see that we have added a few more semantic types and attributes that came from the Car Rental ontology. Namely, we have added `airportOrLocation`, `month`, `day`, `year`, `ratePeriod`, and `divideFactor`.

Overall, we have merged ontologies using the materialized approach. The result contains no redundancies i.e. no concept is represented multiple times in the merged ontology. The overall appearance is as if the ontology was developed right from the beginning with the intention of covering an airfare + car rental domain⁴. This one ontology can now be used for either the airfare or the car rental application or the combination thereof and the two previous ontologies can be scrapped, as is usually the goal when using the materialized approach to ontology merging.

3.1.2 - Virtual Merging through Context Linking

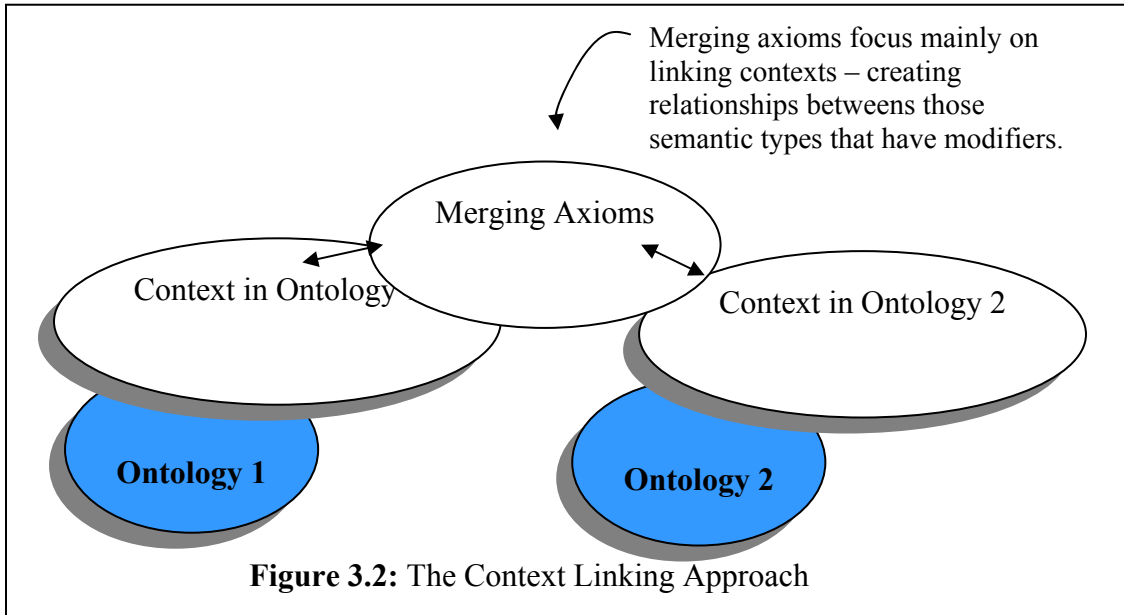
In merging ontologies using the materialized approach, we had to peruse every semantic type and attribute to determine relationships with the other ontology. Meanwhile, virtual merging is concerned with functionality – creating a third application that allows access to the two underlying applications’ sources from one place. The intention is to have the ontologies of the two applications *persist unchanged* (and still be used by their respective ECOIN applications) while the new merged application gives the “virtual” appearance of running on one merged ontology.

Context Linking – Merging is Driven by Context Differences

The context-linking method is used for virtual ontology merging. Rather than actually merging the two ontologies to achieve a new third ontology, we “link the contexts” of the two ontologies. That is, we determine the relationships between those semantic types that can have multiple interpretations. This is sufficient because as we said above, the goal is to achieve an application that allows queries covering multiple ECOIN ontologies – queries that reach sources from both applications (and deal with context issues of both applications). Thus we need to worry only about those semantic types that will be interpreted differently by the two applications – this information is encapsulated in the contexts – hence we call it linking contexts. We need not worry about semantic types that have no modifiers because data that is elevated to those semantic types can be interpreted unambiguously by both applications.

⁴ With the exception, perhaps, of some awkward naming

Thus the main thrust of the merging process is to analyze modifiers and to establish relationships between semantic types that have modifiers. Of course, there are other considerations in merging (for example, flushing out implicit modifiers (see Section 3.2), extending the merged application, etc) but we say that overall, the merging process is *driven* by context differences.



3.1.3 - Linking Contexts Between Airfare and Car Rental

To virtually merge Airfare and Car Rental by linking contexts, we take a look at the semantic types that have modifiers and determine the relationships across the two applications between such semantic types. In essence, we are looking for those semantic types that are equivalent to each other.

Going back to Figure 3.1, we look at `city` from the Car Rental ontology and `cityOrAirport` from the Airfare ontology. `city` has the modifier `airportOrLocation` but `cityOrAirport` has no explicit modifier because Airfare deals only with airport codes. However, we can consider `airportOrLocation` to be an *implicit* modifier of `cityOrAirport`. A modifier can be left implicit in an application if its value would be the same in all contexts of the application (as is the case in Airfare). But when we merge Airfare and Car Rental and consider `city` and `cityOrAirport` equivalent, we must retain the modifier `airportOrLocation` in explicit fashion and assign it the value “airport” for all contexts from the Airfare application. Once `city` and `cityOrAirport` have been merged, the attributes of `destination` and `origin` of semantic type `trip` from (from Airfare), which used to point to `cityOrAirport`, now point to `city` (see Figure 3.3).

We make similar decisions regarding the semantic types `month` and `moneyAmount`, which exist in both applications. In both cases, we realize that the matching semantic types are equivalent. However, `month` from Airfare has an implicit modifier `monthSymType` because that context issue was simply not addressed and the user was left to enter the month symbol in proper format when writing the query. Meanwhile, `currency` is an implicit modifier of `moneyAmount` in Car Rental because all sources are US-based so the value across all contexts is ‘US dollars’. After the merge, however, `currency` is given an explicit value in all the contexts of both applications. Once `month` from Airfare and `month` from Car Rental have been merged and `moneyAmount` from Airfare and `moneyAmount` from Car Rental have been merged, the attribute `hasMonth` of `date` from Airfare points to `month` from Car Rental and those attribute and inheritance arrows that used to point to `moneyAmount` in Car Rental now point to `moneyAmount` in Airfare (see figure 3.3).

The last semantic type with a modifier is `price`. `price` is not equivalent in both applications because `price` in Car Rental has a modifier `ratePeriod` that does not exist for the semantic type `price` from Airfare (nor can `ratePeriod` be considered an implicit modifier of `price` in Airfare). Thus in the merged ontology, we see `price` from both applications have *not* been merged⁵.

Since there are no more semantic types with modifiers (explicit or implicit), we have reached the end of virtual merging through context linking. Below is a picture of what the virtually merged ontology would look like. We note that there are many redundancies in the sense that there are multiple semantic types that represent similar concepts – for example, `date`, `price`, `serviceFee`, etc. However, as explained earlier, this is not a problem because here we are concerned with the practical issue of functionality (allowing source access and context sharing across two applications) and not the theoretical problem of neatly and completely merging two ontologies.

⁵ Two semantic types in an ontology cannot actually have the exact same name and indeed COIN semantic types (and modifiers, etc) are all preceded by a unique application identifier. Thus `price` from Airfare and `price` from Car Rental actually have different names. We leave those unique application prefixes out in the explanation above for the sake of clarity. See section 3.3.2 for more on unique application prefixes.

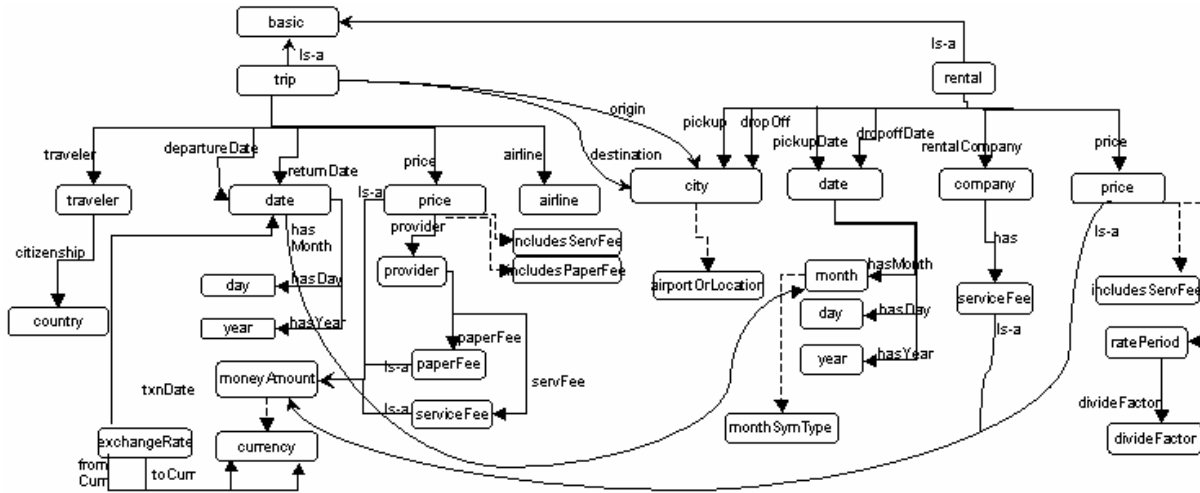


Figure 3.3 – Virtually Merged Ontology

3.2 - How to merge applications – The Merging Algorithm and the Merger Axioms

So far, we have discussed virtual merging through context linking by means of a high level example. The next few sections discuss the technical details of context-linking – they introduce merger axioms, present the merging algorithm, and then go into the details of creating actual merger axioms.

3.2.1 – Merger Axioms

To merge two ECOIN applications through context-linking, a user must create a merger axioms file. These axioms can be created by following the set of steps (a merging algorithm) that we present in more detail in the next section. The purpose of the merger axioms is to allow the abduction engine to reason about the merged application as a legitimate, standalone ECOIN application. The axioms accomplish this by serving three functions (the earlier-stated goals of merging)⁶:

- 1) They bring together the sources of both applications into one merged application
- 2) They reconcile the contexts of the two applications and bring context benefits of both applications into the merged application
- 3) They extend the merged application (i.e. add new semantic types, modifiers, contexts etc).

Of the above three functions, the most important is reconciling contexts. What do we mean by reconciling contexts? We said above that the purpose of the axioms is to allow the abduction engine to reason about the merged applications as one standalone application. When we say

⁶ The goals are stated in slightly different words here to further nuance our understanding of them

“reason about the merged applications” we mean, “reason about the *contexts* of both applications from within one merged application” because, after all, it is context mediation that is the goal of the ECOIN approach. To be able to reason correctly about the contexts of both applications, we must be sure that the contexts make sense. That is, all the modifiers that now exist together within one application must fit correctly into all the contexts that have been inherited from both applications – this is the context reconciliation that the merger axioms perform.

Context reconciliation is achieved by determining isomodifiertypes, isomodifiers, isoattributes and isocontexts between the two applications. These are, in essence, equivalence relationships between the ontology objects of the two applications. Their definitions are as follows:

isomodifiertypes: semantic types that are equivalent with respect to their modifiers. That is, if semantic type A is an isomodifiertype of semantic type B, then for every modifier of semantic type A, there is an equivalent modifier of semantic type B that exists either explicitly or is implicit. By “implicit” we mean the modifier was not declared because it would have the same value for every context in that application. For example, *currency* is implicit in Car Rental because all the sources are domestic and it would have had the value ‘USD’ in every Car Rental context.

isomodifiers: modifiers that are equivalent (even if they have different names)

isoattributes: attributes that are equivalent (even if they have different names)

isocontexts: contexts that are equivalent (even if they have different names)

3.2.2 – How the Merger Axioms Work

For the abduction engine to be able to use the merger axioms, two things are needed: upward inheritance and globally unique names of ontology and context objects. In this section we explain these two concepts and explain how the abduction engine uses the merger axioms.

Upward Inheritance

As discussed in section 1.2.2, the ontology, context information, source information and elevation axioms of an application are described by Prolog rules in a single text file. When merging two applications, these rules are all automatically “inherited upward” into the merger axioms file. That is, simply by declaring that two applications have been merged, the merger axioms file indicates to the abduction engine that all rules from the two underlying application files are to be considered part of the merged application even though they do not explicitly exist in the merger axioms file. Anything new that is declared in the merger axioms file overrides any existing rules from underlying applications.

Unique Naming

The fact that rules originating from multiple applications can exist (albeit implicitly through upward inheritance) within the merged application means that semantic type names, modifier names, etc need to be globally unique because any two applications are potential candidates for merging. If names were not unique, then when the abduction engine looks up a semantic type that happens not to be unique, it would find more than one set of modifiers and would not know how to interpret the data represented by those semantic types. To solve this, ECOIN labels all applications with a globally unique URI. This URI precedes each semantic type/modifier/attribute/context name in an application and furnishing uniqueness in naming.

How the Abduction Engine Uses the Axioms

Overall, the abduction engine treats merged applications as follows [6]: Upon receiving a query from the user, the abduction engine has to determine to which semantic type each of the data columns requested is elevated. First it looks in the merger axioms file. If it finds the elevation information there, it takes it. Otherwise, it determines what two applications have been merged and goes into those application files and looks there for the relevant elevation information. If the elevation rule does not exist there and one (or both) of the underlying applications is also a merged application, the engine then looks into the “grandchildren” applications’ files, and so on, until the appropriate elevation is found.

Upon determining the semantic type, the abduction engine then looks for the modifier information of that semantic type. Once again, it starts in the merger axioms file and if the information is not there, it looks for it through the levels of underlying applications.

In summary, upward inheritance and unique naming allow the abduction engine to treat the merged application as a standalone ECOIN application. Upward inheritance allows the engine access to both applications’ ontology, source and context information, unique naming prevents duplicate name problems and the rest of the merger axioms reconcile the contexts from both applications.

3.2.3 – The Merging Algorithm

So far we have discussed what the merger axioms file is for and how it works. But what is the process through which this merger axioms file is created? To create the merger axioms file, a general set of steps can be followed. The steps are captured in the flow chart in Figure 3.4.

The steps of the algorithm lead the user through the three goals of merging. Looking at figure 3.4, we see that step 1 accomplishes goal 1, steps 2 through 10 accomplish goal 2 and step 11 accomplishes goal 3. Note that the steps related to goal 2 constitute the bulk of the merging algorithm. This is because goal 2 is the most significant function of the merger axioms (as discussed in section 3.3.1).

Each “action item” in the merging flow chart yields one or more merger axioms. Namely, “declaring an isomodifier,” or “pulling up a semantic type,” or “creating a conversion function,” etc are all done by adding axioms to the merger file.

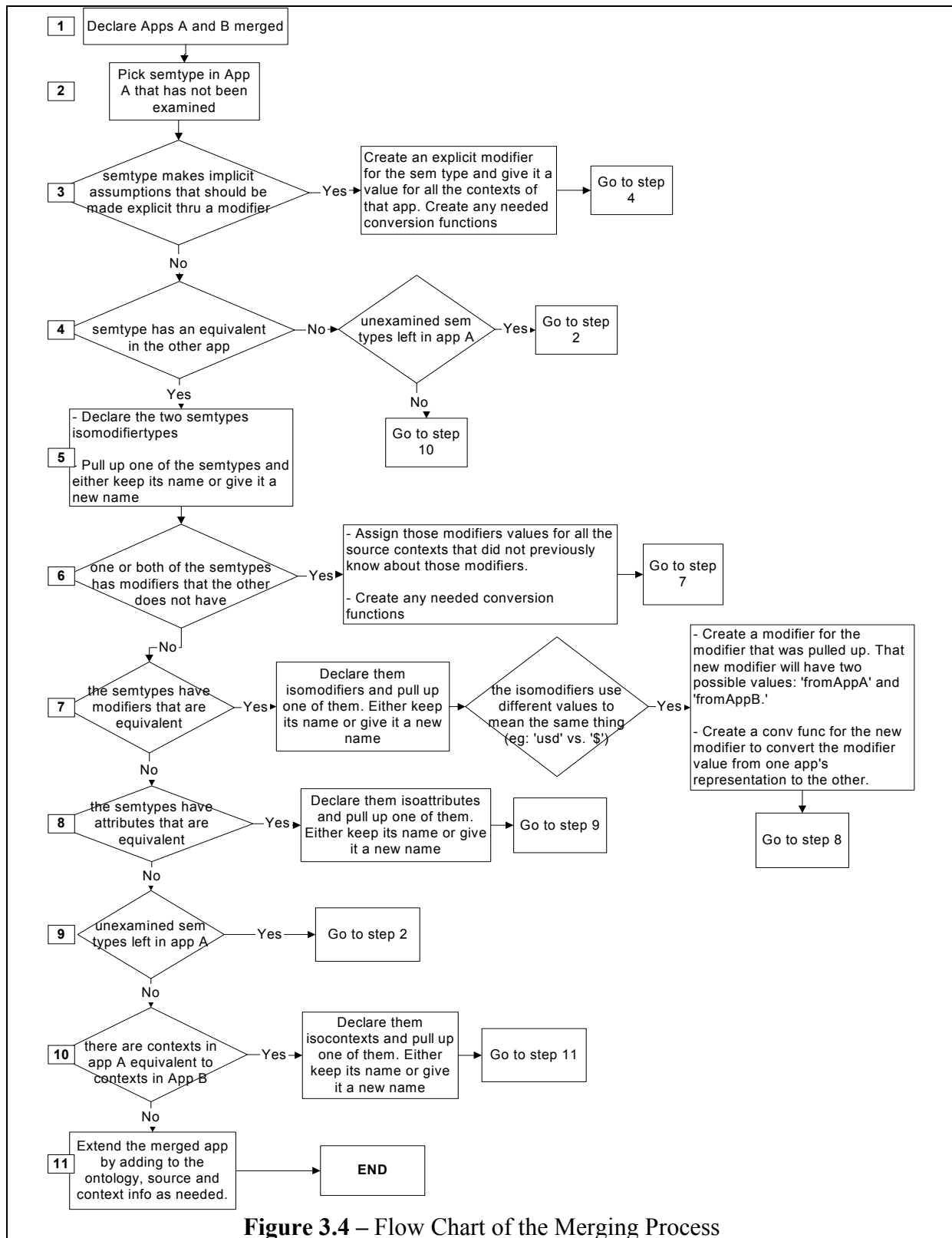


Figure 3.4 – Flow Chart of the Merging Process

3.3 – Understanding the Merger Axioms

The previous section has discussed the steps to create an axioms file. But we still need to understand the actual axioms to the level that we can write them. Accordingly, this section presents the types of merger axioms that accomplish the three goals of merging and the following section discusses the axioms of the Airfare and Car Rental merger in detail.

As discussed in section 2.3.1, there are three goals that can potentially be accomplished when merging any two applications. The merger axioms file must achieve at least Goal 1 and can go on and achieve Goals 2 and 3 as well if the developer so desires. The type of axioms that would be used to achieve the goals are exemplified in the table below:

The Desired Goal	Merger Axioms Required	Example of Axiom in Prolog (the numbering below corresponds to the numbering in the second column)
Seamless access to sources across both applications	1) Declare that the applications have been merged. All existing sources and contexts will be automatically inherited by the new merged application	1) <pre>rule(merges([appAirfare,appCarRental]),(true)).</pre>
Use context capabilities of one application to benefit other application. For ex: want <i>price</i> in Car Rental to obtain currency conversion capability from <i>airfarePrice</i> in Airfare	1) Declare that <code>moneyAmount</code> from Car Rental is equivalent to <code>moneyAmount</code> from Airfare. The appropriate modifiers (i.e. currency) and conversion functions will automatically apply. 2) For each context that merged app inherits from Car Rental, declare a modifier value for currency.	1) <pre>rule(isomodifiertypes (appMergedTravel, appAirfare, price, airfarePrice), (true)).</pre> 2) <pre>rule(modifier(price, O, currency, expediaCarContext, M), (cste(basic, M, expediaCarContext, "USDollar"))).</pre> ... similar rule for the rest of the contexts from Car Rental
Extend merged application with new sources, contexts or modifiers, etc For example, add context <code>FlyAndRent</code> that defines <code>price</code> as $(\text{airfare price}) + (\text{Car Rental Price})$	1) Need axiom for new context 2) New axiom for new modifier being added 3) Need axioms that give that modifier a value in all of the existing contexts and assign all the existing modifiers a value for the new context 4) Need axioms that define conversion functions for the new modifier 5) Need axioms that define new attributes used by the new conv functions that were added	1) <pre>rule(contexts([newContextForFlyAndRent]),(true)).</pre> 2) <pre>rule(modifiers(price, [includesCarRental]), (true)).</pre> 3) <pre>rule(modifier(price, O, includesCarRental, doraContext, M), (cste(basic, Modifier, doraContext, "dontIncludeRental"))).</pre> ... similar rule for the rest of the contexts in the merged app 4) <pre>rule(cvt(commutative, price, O, includesCarRental, Ctxt, "dontIncludeRental", Vs, "yesIncludeRental", Vt), (attr(O, month1, M1), ... plus(airPrice, RentalPrice, Result))).</pre> 5) <pre>rule(attr(Price, month1, Mnth1), (yahoo_p(Price, Mnth1))).</pre>

Table 3.1 – Summary of Merger Axioms Table

3.4 – The Merger Axioms of Airfare + Car Rental

In this section, we work our way through the key merger axioms of the Travel Application (the merger of Airfare and Car Rental) as an example that explains how to read and create merger axioms (Appendix C contains the full merger axioms file). This example will elucidate the details that lie between the lines of the table above. After this section, the table above can be used as a useful summary reference. Furthermore, the merger axioms discussed in this section can all be automatically generated by the proposed CLAMP tool, based on input from the user regarding the merging decisions. We discuss this in detail in Chapter 4.

Seamless access to sources from both applications:

The first axiom declares:

```
rule(merges([application512,application513]),(true)).
```

The first thing to note is the application numbers – ECOIN refers to applications by a unique number. From hereon, we note that 512 refers to Airfare, 513 to Car Rental and 514 refers to the merged application (as represented by the merger axioms file).

The first axiom declares that applications 512 (Airfare) and 513 (Car Rental) have been merged. That one rule causes the merged application to “upwardly inherit” all the ontology, context, source and elevation information from both underlying applications. The rule enables the user to query sources from both underlying applications through one seamless query in the merged application.

This type of rule (i.e. the rule that allows seamless access to sources) is summarized in the first row of the table above.

Cross fertilization of contexts:

The next axiom is as follows:

```
rule(isomodifiertypes(application514,application512,moneyAmount2,moneyAmount),
      (true)).
```

This rule would be read as follows in plain English:

The semantic type, *moneyAmount2* (from Car Rental) and *moneyAmount* (from Airfare) are *isomodifiertypes* i.e. they are equivalent because they have the same set of modifiers. Thus “pull up” *moneyAmount2* into the merged application, i.e. use the name, *moneyAmount2*, to refer to this semantic type in the emerged application.

By declaring the two semantic types to be equivalent with respect to their modifiers, the axiom is saying that any modifier of those two semantic types that may be found in the airfare or car rental application is now to be used in the new, merged application. *MoneyAmount2* (from car rental) has no notion of *currency* (or one can consider it implicit). But since it has been declared an “*isomodifiertype*” of *moneyAmount* (from Airfare), it automatically inherits the *currency* modifier (and associated conversion functions). This axiom thus allows the merged application to apply currency conversion capabilities of the airfare application upon data coming from sources inherited from the Car Rental application.

However, now that `moneyAmount2` from Car Rental has inherited the currency modifier, all the contexts that were inherited from Car Rental need to be assigned a currency value since they previously had no such notion. The following rules perform that function:

```
rule(modifier(moneyAmount2, O, currency, joe, M),
      (cste(basic, M, joe, "USD"))).
rule(modifier(moneyAmount2, O, currency, c_expediacar, M),
      (cste(basic, M, c_expediacar, "USD"))).
rule(modifier(moneyAmount2, O, currency, c_yahoocar, M),
      (cste(basic, M, c_yahoocar, "USD"))).
rule(modifier(moneyAmount2, O, currency, c_qixocar, M),
      (cste(basic, M, c_qixocar, "USD"))).
```

These four rules assign the currency value “USD” for the four contexts inherited from Car Rental (`joe`, `c_expediacar`, `c_yahoocar`, `c_qixocar`).

There are three more `isomodifiertype` rules declared in the merger file:

```
rule(isomodifiertypes(application514,application513,cityORAirport,city), (true)).
rule(isomodifiertypes(application514,application513,month,month2), (true))
rule(isomodifiertypes(application514,application513,day,day2), (true)).
```

The first rule declares `city` from Car Rental to be equivalent to `cityOrAirport` from Airfare and “pulls up” `cityOrAirport` – i.e. the merged application will use the name `cityOrAirport`. Now, `cityOrAirport` inherited from Airfare, has the airport code to city name conversion capability that `city` from Car Rental has. The second rule declares `month2` from Car Rental to be equivalent to `month` from Airfare and pulls up `month`. Now `month`, inherited from Airfare, has the month symbol conversion capability that `month2` from Car Rental has. The third rule declares `day2` from Car Rental to be equivalent to `day` from Airfare and pulls up `day`.

Similar to the case above with `moneyAmount` and `currency`, now that `month` and `cityOrAirport` from Airfare have inherited the `airportOrLocation` and `monthSymType` modifiers, all the contexts that were inherited from Airfare need to be assigned modifier values for those two modifiers. The following rules perform that function:

```
rule(modifier(cityORAirport, O, airportOrLocation, dora, M),
      (cste(basic, M, dora, "location"))).
rule(modifier(cityORAirport, O, airportOrLocation, doras_friend, M),
      (cste(basic, M, doras_friend, "location"))).
rule(modifier(cityORAirport, O, airportOrLocation, c_yahoo, M),
      (cste(basic, M, c_yahoo, "airport"))).
rule(modifier(cityORAirport, O, airportOrLocation, c_expedia, M),
      (cste(basic, M, c_expedia, "airport"))).
rule(modifier(cityORAirport, O, airportOrLocation, c_orbitz, M),
      (cste(basic, M, c_orbitz, "airport"))).
rule(modifier(cityORAirport, O, airportOrLocation, c_itn, M),
      (cste(basic, M, c_itn, "airport"))).
rule(modifier(cityORAirport, O, airportOrLocation, c_travelselect, M),
      (cste(basic, M, c_travelselect, "airport"))).

rule(modifier(month, O, month2SymbolType, dora, M),
      (cste(basic, M, dora, "threeLetter"))).
rule(modifier(month, O, month2SymbolType, doras_friend, M),
```

```

        (cste(basic, M, doras_friend, "numeric"))).
rule(modifier(month, O, month2SymbolType, c_yahoo, M),
      (cste(basic, M, c_yahoo, "threeLetter"))).
rule(modifier(month, O, month2SymbolType, c_expedia, M),
      (cste(basic, M, c_expedia, "numeric"))).
rule(modifier(month, O, month2SymbolType, c_orbitz, M),
      (cste(basic, M, c_orbitz, "threeLetter"))).
rule(modifier(month, O, month2SymbolType, c_itn, M),
      (cste(basic, M, c_itn, "threeLetter"))).
rule(modifier(month, O, month2SymbolType, c_travelselect, M),
      (cste(basic, M, c_travelselect, "numeric"))).

```

The set of merger axioms just discussed (i.e. isomodifiertypes, assigning modifier values to newly inherited modifiers) is what allows cross fertilization of contexts – use of the context capabilities of one application to benefit the other application. This set of rules is summarized in the second row of Table 3.1.

Extending the merged application

The next set of merger axioms begins with:

```

rule(contexts([c_FlyAndRent]), (true)).

rule(modifiers(price, [includesCarRental]), (true)).
rule(modifiers(provider, [includesCarCompany]), (true)).

```

The first rule introduces a new context, `c_FlyAndRent`, into the merged application. The purpose of this context is to extend the merged application to provide benefits that cannot be achieved by simply extending one (or both) of the underlying applications. This context enriches the number of meanings of `price` and of `provider`. Namely, the next two rules bestow a new modifier upon `price` and `provider`. Previously `price` referred to airfare price and the semantic possibilities ranged from including service fees to paper ticket charges to visa fees. Now a new modifier for `price`, `includesCarRental`, is being declared and will have the value “yes” in the `c_FlyAndRent` context. Thus a user can now set his context to `c_FlyAndRent` so that the price quoted to him includes a car rental at his destination. Similarly, a new modifier for `provider`, `includesCarCompany` is being declared and will have the value “yes” in the `c_FlyAndRent` context. Thus a user can know the car rental company as well as the airfare provider (i.e. Orbitz, Yahoo, etc) when he is quoted the combined price in the `c_FlyAndRent` context.

Now that a new context has been introduced, all the modifiers that exist in the merged application need to be assigned values in the new context:

```

rule(modifier(price, Object, includesCarRental, c_FlyAndRent, Modifier),
      (cste(basic, Modifier, c_FlyAndRent, "yes"))).
rule(modifier(price, O, includesServFee, c_FlyAndRent, M),
      (cste(basic, M, c_FlyAndRent, "yes"))).
rule(modifier(price, O, includesVisaFee, c_FlyAndRent, M),
      (cste(basic, M, c_FlyAndRent, "no"))).
rule(modifier(price, O, includesPaperCharge, c_FlyAndRent, M),
      (cste(basic, M, c_FlyAndRent, "no"))).

```

```

rule(modifier(price2, Object, includesServiceFee, c_FlyAndRent, M),
      (cste(basic, M, c_FlyAndRent, "yes"))).
rule(modifier(cityORAirport, O, airportOrLocation, c_FlyAndRent, M),
      (cste(basic, M, c_FlyAndRent, "airport"))).
rule(modifier(month, O, month2SymbolType, c_FlyAndRent, M),
      (cste(basic, M, c_FlyAndRent, "threeLetter"))).
rule(modifier(moneyAmount2, O, currency, c_FlyAndRent, M),
      (cste(basic, M, c_FlyAndRent, "USD"))).
rule(modifier(provider, Object, includesCarCompany, c_FlyAndRent, M),
      (cste(basic, M, c_FlyAndRent, "yes"))).

```

Furthermore, the two new modifiers that were introduced need to be assigned values for all the already existing contexts:

```

rule(modifier(price, Object, includesCarRental, dora, Modifier),
      (cste(basic, Modifier, dora, "no"))).
rule(modifier(price, Object, includesCarRental, doras_friend, Modifier),
      (cste(basic, Modifier, doras_friend, "no"))).
rule(modifier(price, Object, includesCarRental, c_expedia, Modifier),
      (cste(basic, Modifier, c_expedia, "no"))).
rule(modifier(price, Object, includesCarRental, c_yahoo, Modifier),
      (cste(basic, Modifier, c_yahoo, "no"))).
rule(modifier(price, Object, includesCarRental, c_itn, Modifier),
      (cste(basic, Modifier, c_itn, "no"))).
rule(modifier(price, Object, includesCarRental, c_orbitz, Modifier),
      (cste(basic, Modifier, c_orbitz, "no"))).
rule(modifier(price, Object, includesCarRental, c_travelselect, Modifier),
      (cste(basic, Modifier, c_travelselect, "no"))).

rule(modifier(provider, O, includesCarCompany, dora, M),
      (cste(basic, M, dora, "no"))).
rule(modifier(provider, O, includesCarCompany, doras_friend, M),
      (cste(basic, M, doras_friend, "no"))).
rule(modifier(provider, O, includesCarCompany, c_yahoo, M),
      (cste(basic, M, c_yahoo, "no"))).
rule(modifier(provider, O, includesCarCompany, c_expedia, M),
      (cste(basic, M, c_expedia, "no"))).
rule(modifier(provider, O, includesCarCompany, c_orbitz, M),
      (cste(basic, M, c_orbitz, "no"))).
rule(modifier(provider, O, includesCarCompany, c_itn, M),
      (cste(basic, M, c_itn, "no"))).
rule(modifier(provider, O, includesCarCompany, c_travelselect, M),
      (cste(basic, M, c_travelselect, "no"))).

```

Finally, new conversion functions have to be written for the two new modifiers:

```

rule(cvt(commutative, price, O, includesCarRental, Ctxt, "no", Vs, "yes", Vt),
      (yahoocar_p(DestC, Dropoff, M1C, M2C, D1C, D2C, Rp, _, _),
       attr(O, month1, M1),
       attr(O, month2, M2),
       attr(O, day1, D1),
       attr(O, day2, D2),
       attr(O, destination, Dest),
       value(M1, c_yahoo, M1v),
       value(M1C, c_yahoo, M1v),
       value(M2, c_yahoo, M2v),
       value(M2C, c_yahoo, M2v),
       value(D1, c_yahoo, D1v),
       value(D1C, c_yahoo, D1v),
       value(D2, c_yahoo, D2v),
       value(D2C, c_yahoo, D2v),
       value(Dest, c_yahoo, Destv),

```

```

value(DestC, c_yahoo, Destv),
value(Dropoff, c_yahoo, "same"),
value(Rp, Ctxt, Rpv),
plus(Vs, Rpv, Vt)).

rule(cvt(commutative, provider, P, includeCarCmpny, Cxt, "no", Vs, "yes", Vt),
(yahoocar_p(DestC, Dropoff, M1C, M2C, D1C, D2C, _, Comp, _),
attr(O, provider, P),
attr(O, month1, M1),
attr(O, month2, M2),
attr(O, day1, D1),
attr(O, day2, D2),
attr(O, destination, Dest),
value(M1, c_yahoo, M1v),
value(M1C, c_yahoo, M1v),
value(M2, c_yahoo, M2v),
value(M2C, c_yahoo, M2v),
value(D1, c_yahoo, D1v),
value(D1C, c_yahoo, D1v),
value(D2, c_yahoo, D2v),
value(D2C, c_yahoo, D2v),
value(Dest, c_yahoo, Destv),
value(DestC, c_yahoo, Destv),
value(Dropoff, c_yahoo, "same"),
value(Comp, Ctxt, Compv),
concat(Vs, "&", Vt1),
concat(Vt1, Compv, Vt))).

```

The first `includesCarRental` conversion function works as follows. It is given an airfare price. It then determines the dates that the user wants to arrive at and depart and it determines the destination. It then sends a query to the yahoo car rental aggregator giving the destination and dates it has just determined. Finally, it adds the car rental rate to the airfare price.

The `includesCarCompany` conversion function works in essentially the same way as the `includesCarRental` conversion function. The only difference is that instead of determining the car rental price, it determines the car rental company offering the car rental price (i.e. Budget, Hertz, etc) and then it returns both the car rental company and the airfare provider so that user can know what combination of airfare provider and car rental company is offering the combined price that he sees.

Finally, we note that the two conversion functions use several `attr()` functions. These functions allow the abduction engine to glean some needed piece of data by defining where in the data source schema a certain piece of data exists. For example, we mentioned above that the conversion function for `includesCarRental` determines the dates that the user wants to travel. In order to do that, one of the things that needs to be looked up is the month the user wants to depart. So the conversion function uses the following `attr` function: `attr(O, month1, M1)`. The first element within the parentheses, `O`, represents the price object. The second element, `month1`, represents the name of the `attr` relationship. And the third element, `M1`, is to hold the month value once it is looked up. But how will it lookup the month? The `attr(X, month1, Y)` function needs to be defined elsewhere such that the abduction engine knows where in the source schema the `month1` attribute exists. This is achieved by rules of the following format:

```
rule(attr(Price, month1, Mnth1), (yahoo_p(_, Price, _, _, Mnth1, _, _, _, _))).
```

This rule defines the month1 attribute by showing the respective positioning of price and month1 within the yahoo schema⁷: (represented by yahoo_p(.,...,)). Namely, Price is in the second column and Mnth1 is in the fifth column. Since the conversion function is given price, it can then use this “respective positioning” information to find month1.

All the attr (functions that are used within the conversion functions have to either already exist within the underlying application files or they need to be defined in the merger axioms file. Thus there exist the following set of attr (definitions within the merger axioms to define those functions that do not already exist in the underlying applications’ files:

```
rule(attributes(price, [month1, month2, day1, day2, destination]), (true)).

rule(attr(Price, month1, Mnth1), (myorbitz_p(_, Price, _, _, Mnth1, _, _, _, _))).
rule(attr(Price, month1, Mnth1), (yahoo(_, Price, _, _, Mnth1, _, _, _, _))).

rule(attr(Price, date1, Dt1), (expedia2_p(_, Price, _, _, Dt1, _, _, _))).
rule(attr(Price, date1, Dt1), (expedia_p(_, Price, _, _, Dt1, _, _, _))).

rule(attr(Price, month2, Mnth2), (myorbitz_p(_, Price, _, _, Mnth2, _, _, _, _))).
rule(attr(Price, month2, Mnth2), (yahoo(_, Price, _, _, Mnth2, _, _, _, _))).
rule(attr(Price, date2, Dt2), (expedia2_p(_, Price, _, _, Dt2, _, _, _))).
rule(attr(Price, date2, Dt2), (expedia_p(_, Price, _, _, Dt2, _, _, _))).

rule(attr(Price, day1, Dy1), (myorbitz_p(_, Price, _, _, Dy1, _, _, _))).
rule(attr(Price, day1, Dy1), (yahoo(_, Price, _, _, Dy1, _, _, _))).

rule(attr(Price, day2, Dy2), (myorbitz_p(_, Price, _, _, Dy2, _, _, _))).
rule(attr(Price, day2, Dy2), (yahoo(_, Price, _, _, Dy2, _, _, _))).

rule(attr(Price, destination, Destination),
      (myorbitz_p(_, Price, Destination, _, _, _, _))).
rule(attr(Price, destination, Destination),
      (yahoo(_, Price, Destination, _, _, _, _))).
rule(attr(Price, destination, Destination),
      (expedia2_p(_, Price, Destination, _, _, _, _))).
rule(attr(Price, destination, Destination),
      (expedia_p(_, Price, Destination, _, _, _, _))).

rule(attr(Price, destination, Destination),
      (yahoo(_, Price, Destination, _, _, _, _))).
rule(attr(Price, destination, Destination),
      (expedia2_p(_, Price, Destination, _, _, _, _))).
rule(attr(Price, destination, Destination),
      (expedia_p(_, Price, Destination, _, _, _, _))).
```

The set of merger axioms just discussed (i.e. introducing a new context, new modifiers, new attributes, new conversion functions) all fall under the merging goal of extending the application to provide benefits that cannot be achieved by simply extending one (or both) of the underlying applications. This set of rules is summarized in the third row of Table 3.1.

⁷ Yahoo is a website and has no “schema.” But when we say the yahoo source, we mean data from Yahoo after it has been extracted by Cameleon and modeled as a database table. This allows us to speak of “yahoo’s schema.”

Ismodifiers, isocontexts and isoattributes:

Finally, we note that while the Airfare and Car Renter merger does not provide such an example, the merger axioms also allow the declaration of “isomodifiers”, “isocontexts” and “isoattributes”. These are analogous to isomodifiertypes – they indicate equivalence and pull up one of the two objects to the merged application (see Section 3.2.1 for definitions).

3.5 Sample Run of Merged Application

Part of the reason behind merging the Airfare and Car Rental applications is to present a case study that demonstrates the viability of merging in accomplishing the three goals it sets for itself (see section 2.3.1). To complete the example, let us actually run queries in the merged application.

3.5.1 – Seamless Access to Sources

Below we see a single, seamless query asking data from a source from the Airfare application, yahoo and a source from the Car Rental Application, expediacar. The query is basically searching for a fare from Boston to San Francisco and for car rental rates for the duration of the stay:

SQL	<pre>select yahoo.Destination, yahoo.Departure, yahoo.Airline, yahoo.Price, expediacar.Company, expediacar.Price, expediacar.Rateperiod from yahoo, expediacar where yahoo.Destination="SFO" and yahoo.Departure="BOS" and yahoo.Month1="Aug" and yahoo.Month2="Aug" and yahoo.Day1="15" and yahoo.Day2="30" and expediacar.Pickup="SFO" and expediacar.Dropoff="" and expediacar.Date1="08/15/03" and</pre>
Context	none <input type="button" value="v"/>

Figure 3.5 – Query showing access to sources from both underlying apps in merged app

The result that is returned is below. We see with one query, we were able to determine an airfare to San Francisco and also various rental rates in San Francisco for the duration of the stay.




yahoo.Destination	yahoo.Departure	yahoo.Airline	yahoo.Price	expediacar.Company	expediacar.Price	expediacar.Rateperiod
SFO	BOS	 ATA	390	National	147.99	Week
SFO	BOS	 ATA	390	Budget	157.99	Week
SFO	BOS	 ATA	390	Hertz	158.99	Week

Figure 3.6 – Result of Query to sources from both underlying apps in merged application

3.5.2 – Cross Fertilization of Contexts

The query below is to a source from the Airfare application – Travelexport (a UK source). Remember that Airfare has no notion of airport to city name conversion nor of month symbol type conversion. But the merged application inherited these context capabilities from Car Renter. Thus the query below can enter locations by the city name rather than airport code and the month is entered as a three letter abbreviation rather than numerically:

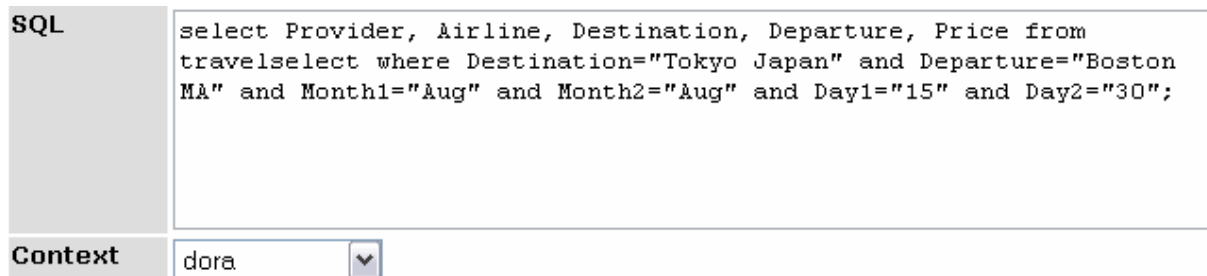


Figure 3.7 – Query showing cross fertilization of contexts in merged application

The conflict detection table (Figure 3.7) that is generated shows context mediation ability inherited from both applications – month symbol and airport code conversion from Car Renter and currency, service fees and paper ticket charge from Airfare. For example, if we look at the first row of the table, we see the semantic type `price` in the first column. This is a semantic type that originated in Airfare. Moving down the row to the fourth column, we see that a conflict is being reported for the `currency` modifier of `price`. The next two columns show the details of the conflict: the `currency` value is `GBP` in `c_travelexport` context and `USD` in `dora` context.

Nothing extraordinary has happened so far because `currency` has always been a part of the Airfare application. But if we look three rows down in the same table, we see `month`, which originated in Airfare showing a conflict for its `month2SymbolType` modifier. But Airfare has no such modifier – the merged application inherited it from Car Rental. Context cross-fertilization has been achieved.

SemanticType	Column	Source	Modifier	Modifier value in source context	Modifier value in target context	Conversion Function
price	IV	travelselect (Name, C2, C4, C4, C6, C6, 15, 30, Name, C10)	currency	c_travelselect : GBP	dora : USD	olsen_p(V12, V11, V10, V9), value(V12, V8, V7), value(V11, V8, V6), value(V10, V8, V5), currentDate_p(V4), value(V4, V8, V3), value(V9, V8, V3), multiply(V2, V5, V1)
price	IV	travelselect (Name, C2, C4, C4, C6, C6, 15, 30, Name, C10)	includesPaperCharge	c_travelselect : no	dora : yes	attr(V7, provider, V6), attr(V6, paperTicketFee, V5), value(V5, V4, V3), plus(V2, V3, V1)
price	C2	travelselect (Name, C2, C4, C4, C6, C6, 15, 30, Name, C10)	includesServFee	c_travelselect : no	dora : yes	attr(V7, provider, V6), attr(V6, serviceFee, V5), value(V5, V4, V3), plus(V2, V3, V1)
month	C6	travelselect (Name, C2, C4, C4, C6, C6, 15, 30, Name, C10)	month2SymbolType	c_travelselect : numeric	dora : threeLetter	month_symbol_converter(V2, V1)
cityORAirport	C4	travelselect (Name, C2, C4, C4, C6, C6, 15, 30, Name, C10)	airportOrLocation	c_travelselect : airport	dora : location	airport_code_lookup(V2, V1)

Figure 3.8 – Conflict detection for query showing cross fertilization of contexts

The result of the query is as follows:

Provider	Airline	Destination	Departure	Price
TravelSelect	Continental Airlines 0843	Tokyo Japan	Boston MA	1284.50411

Figure 3.9 – Results of Query showing cross fertilization of contexts in merged application

3.5.3 – Value Added Benefits

A new context, `c_FlyAndRent` has been introduced into the merged application. Two new modifiers, `includesCarRental` and `includesRentalCompany` have been defined for the semantic types `price` and `provider`. Thus the query below, which asks for Price from yahoo will return with a number that is the sum of the airfare as well as the car rental at the destination for the duration of the stay. Also the provider that is returned will be the Airfare provider as well as the car rental company.

SQL	<pre>select Provider, Airline, Destination, Departure, Price from yahoo where Destination="SFO" and Departure="BOS" and Month1="Aug" and Month2="Aug" and Day1="15" and Day2="30";</pre>
Context	c_FlyAndRent ▾

Figure 3.10 – Query showing value-added benefits in merged application

We see in the generated SQL that the two prices are being added and that the airfare provider and the rental company are being concatenated:

```
select ((yahoo.Provider||'&')||yahoocar.Company), yahoo.Airline, 'SFO', 'BOS',
(yahoo.Price+yahoocar.Price)
from (select Airline, Price, 'SFO', 'BOS', 'Aug', 'Aug', '15', '30', Provider, IsIn
from yahoo
where Destination='SFO'
and Departure='BOS'
and Month1='Aug'
and Month2='Aug'
and Day1='15'
and Day2='30') yahoo,
(select 'SFO', 'same', 'Aug', 'Aug', '15', '30', Price, Company, Rateperiod
from yahoocar
where Pickup='SFO'
and Dropoff='same'
and Month1='Aug'
and Month2='Aug'
and Day1='15'
and Day2='30') yahoocar
```

The result of the query is as follows:

Provider	Airline	Destination	Departure	Price
Yahoo&PAYLESS	 ATA	SFO	BOS	528.88
Yahoo&DOLLAR	 ATA	SFO	BOS	548.99
Yahoo&ALAMO	 ATA	SFO	BOS	550.95
Yahoo&THRIFTY	 ATA	SFO	BOS	552.99

Figure 3.11 – Results of query showing cross fertilization of contexts in merged application

3.6 – Complete List of Merging Capabilities

Thus far, through the example of the Airfare+Car Renter merger, we have demonstrated various merging capabilities. However, the example does not demonstrate all the merging capabilities because it is difficult to find such an artificially comprehensive example in real life. Thus, for the sake of completeness, this section lists all the capabilities of merging, including those not demonstrated.

The capabilities fall under the three merging goals that we have repeatedly emphasized. We demonstrated Goal 1 completely:

- 1) Bringing together sources from multiple applications into one application

We demonstrated Goal 3 almost completely:

- 1) Extending the application by adding a new context, modifiers and attributes
- 2) One can similarly add a new source, new semantic types and elevate the source to the new semantic types (or elevate an existing source to the new semantic types).

Goal 2 contains the most significant function of the merger axioms – reconciling contexts. There is much that can be done in the way of reconciling contexts thus one merging case study cannot cover it. Thus below we list all the possible context reconciliation scenarios between two applications and describe how they would be solved using merger axioms. In the list below we refer to an application A and an application B that are being merged.

SemTypeA is a semantic type in Application A and semTypeB is from Application B.

- 1) SemTypeA and SemTypeB are equivalent and have an identical set of modifiers.
Solution: Declare semTypeA and semTypeB isomodifiertypes and pull one of them up (i.e. indicate which of the two semantic type names should be used in the merged application). Also, declare the modifiers of semTypeA to be the isomodifiers of semTypeB's modifiers and choose which modifiers to pull up.

Example: Say `air.moneyAmount` and `car.moneyAmount` both already have a `currency` modifier. Then `air.moneyAmount` and `car.moneyAmount` should be declared isomodifiertypes and one of them should be pulled up. Also, `air.currency` and `car.currency` should be declared isomodifiers and one of them is pulled up.

- 2) A semantic type from Application A has the same name as a semantic type from Application B but the two actually describe unrelated concepts.
Solution: Do nothing because the unique application URIs that are appended to each semantic type solve the namespace problem
- 3) SemTypeA and semTypeB represent the same concept (i.e. they are semantically equivalent) but have a differing number of modifiers or have modifiers that are not semantically equivalent.
Solution: This is a case of implicit modifiers (see section 3.1.3). Declare semTypeA and semTypeB isomodifiertypes – and pull up one of the two semTypes. The modifiers from both applications will automatically be available in the merged

application as the modifier set of the one semantic type that was pulled up.

Example: Say `air.moneyAmount` has modifiers `currency` and `includesServFee` and `car.moneyAmount` has no modifiers. Then `air.moneyAmount` and `car.moneyAmount` should be declared `isomodifiertypes` and one of them should be pulled up. Suppose `car.Amount` was pulled up – it would automatically have `currency` and `includesServFee` as its modifiers in the merged application.

- 4) `SemTypeA` and `semTypeB` are semantically equivalent and they have the same number of modifiers, which are also semantically equivalent. But they have modifier *values* that are different.

Solution: Declare them `isomodifiertypes`. Any of the modifier values can now be used by declaring context definitions accordingly. However, new conversion functions may need to be defined to convert from a modifier value inherited from one application to a modifier value inherited from another application.

- 5) A modifier from Application A is equivalent to a modifier from Application B but they use different modifier values to refer to the same thing. For example, Application A has a modifier `currency`, a possible value of which is ‘USD’. Application B also has `currency` but it uses ‘USDol.’

Solution: Declare the two `isomodifiers`. Next create a modifier, `currencySymType` for the modifier `currency`. `currencySymType` will have two possible values: ‘fromAppA’ and ‘fromAppB.’ The conversion function for `currencySymType` will convert `currency` values from Application A’s representation to Application B’s representation. So for example, if this conversion were to be done through a table lookup, the table would show that ‘USD’ is equivalent to ‘USDol’.

This solution works for dynamic modifier values as well (i.e. values that are determined dynamically during runtime) because the modifier value conversion is done after the modifier value has already been determined. So, in this example, say `currency` is a dynamic modifier. Then Application A will first dynamically figure out that its value is ‘USD’ and then realize that `currency` itself has a modifier and convert ‘USD’ to ‘USDol’.

3.7 – Chapter Summary – ECOIN Application Merging in a Nutshell

This chapter has taken us through a detailed exposition of merging: the goals, the theory behind it, types of merging, how it is done, and a detailed example. We refer the reader to Appendix E for an abridged “demo” of the two applications, the motivations to build them, the motivation for merging, the process of merging and a demo of the merged application. As a summary we emphasize the main points of this chapter in order to bring all the key concepts together in one place.

The traditional approach to merging ontologies can be termed materialized merging – where two ontologies are brought together to create a third ontology, normally with the intention of scrapping the two constituent ontologies. Meanwhile, virtual merging is concerned with functionality – creating a third application that allows access to the two merged applications’ sources from one place. The intention is to have the ontologies of the two applications *persist unchanged* while the new merged application gives the “virtual” appearance of one ontology.

Context Linking is a type of virtual merging that is driven by context differences – that is, we only worry about those parts of the ontologies that are affected by context differences.

Merging applications has three goals/benefits:

- 1) Seamless Access to sources from both applications from within one application
 - 2) Cross fertilization of contexts: using the context capabilities of one application to benefit the other application
 - 3) Value Added Benefits (extending the merged application to add benefits that would not be possible to achieve even by extending the underlying applications on their own)
- merging has three potential goals

To actually merge two ECOIN applications, a merger axioms file has to be created. The purpose of the merger axioms is to allow the abduction engine to reason about the merged application as a legitimate, standalone ECOIN application. At the very least, it contains an axiom that allows access to sources from both of the underlying applications – Goal 1 above. The developer can optionally add additional axioms that accomplish the remaining two goals.

The merger axioms file works through upward inheritance – all information from the underlying application files is automatically inherited into the merger axioms file. Anything additional that is written into the axioms (to reconcile the contexts, for example) will either override what has been upwardly inherited or, if it is new information, will extend the application.

In the end, the merged application is just another ECOIN application. What is inside the abstraction, i.e. how its context and ontology information is represented has changed but users access it as any other ECOIN application and can merge it yet again with some other application and so on to any arbitrary level.

4. CLAMP – A tool to facilitate Application Merging

To this point, we have discussed, in detail, the application merging process on the ECOIN system. One of the key benefits of application merging is that it significantly cuts the time it would otherwise take to create an application that provides access to the sources and the context capabilities of two existing applications. Namely, without context-linking, a developer would have to create a new application “from scratch”, develop an ontology that covers both domains, elevate all the sources to this new ontology, define contexts and so on. However, context linking provides complete, automatic ontology, context and other application code re-use by means of upward inheritance and time only need be spent on writing a merger axioms file that reconciles the contexts from both applications.

However, writing merger axioms by hand in Prolog is potentially a tedious and bug-prone process. In fact, writing the axioms by hand is not ideal for three reasons:

- 1) Most developers are not proficient at Prolog. Even though the application files and the merger axioms do not require a very detailed knowledge of Prolog, it is still an obstacle for developers.
- 2) Writing the merger axioms by hand is repetitive, error-prone and time-consuming. Representing the ontology and context-information in Prolog is repetitive because the developer needs to write rule after rule for each semantic-type, modifier, attribute, elevation, source, etc. Since he is writing these by hand, the process is time-consuming and prone to small errors.
- 3) Prolog does not have good development and debugging support tools. Thus it is hard to pinpoint where an error is and yet these small syntax errors that inevitably crop up when written by hand can cause major problems. There *are* a couple of Prolog compilers and debuggers (for example, Eclipse) that allow the developer to step through the call stack of the Prolog code execution until the error is found. But these would be an additional piece of technical software that a developer would have to learn.

Overall, we see that while application merging through context-linking greatly reduces the amount of work needed to build an application that covers two existing application domains, the context-linking process can be intimidating or at least slower and more-error prone than it should be. Thus, in this chapter we propose the design of an application-merging tool, CLAMP, to ameliorate the three problems of: (1) developers needing to know Prolog, (2) creation of merger axioms being slow and repetitive, and (3) the axioms being error-prone. Addressing these problems would make application merging much easier and thus viable for wide use.

4.1 - What is CLAMP?

CLAMP stands for Context-Linking Application Merging Process. The purpose of the CLAMP tool is to facilitate merger axiom creation for a user that understands the ECOIN approach to context mediation and context linking (even if not the technical details of writing merger axioms). It is not a tool that takes two ECOIN applications and automatically merges them.

To be more specific, CLAMP takes the two applications to be merged and then prompts the user, step-by-step, for the information needed to reconcile the contexts and extend the merged application. It then generates the merger axioms that reflect all the decisions made by the user. The key point is that the user makes all the merging decisions while CLAMP provides the relevant information that is helpful in making the decisions, then implements those decisions in Prolog.

The next section details the assumptions made about the user who will be using CLAMP.

4.2 - What assumptions are made about the user?

The following assumptions are made about the user of the CLAMP tool.

- 1) Knowledgeable of the ECOIN approach to context mediation. This is so that the user has the ability to understand the applications that are about to be merged so that he can understand the context issues the applications address. Thus he needs to know the theory behind ECOIN applications and the ECOIN model (i.e. ontology, context, and source information).
- 2) (not essential) Knows how to build a ECOIN application using the Application Editor tool⁸. Knowing Application Editor is required for two reasons – (a) the CLAMP tool’s layout and design is similar to the application editor and so it would make usage of CLAMP easier and (b) One potential aspect of merging is that the developer might want to extend the merged application by adding a new context, source, semantic type, etc. To do this, CLAMP will actually forward the user to Application Editor because merged applications are conceptually no different from any other ECOIN application and so Application Editor should be used to “edit” the application.
- 3) Understands the ECOIN model details of each of the applications he wants to merge. This is necessary because the user will be prompted to make all the decisions during merging. He will need to know which semantic types and modifiers are relevant in

⁸ The Application Editor, discussed in full detail in [cite Phil’s thesis] allows users to develop and edit COIN applications without having to enter Prolog code. The user simply needs to enter the ontology (semantic types, modifiers, attributes) the sources, elevations, contexts and conversion functions and the tool generates the application file in various representations (RDF, RuleML, RFML, Prolog).

resolving the context differences between the two applications. He will need to be able to spot implicit modifiers, declare isomodifiertypes and so on.

4.3 - The Tool's Approach

The CLAMP tool thinks of the merging process as consisting of multiple stages and its approach is to walk the user through these stages prompting for the relevant information at each stage. These stages roughly correspond to the three goals of merging discussed earlier. We mention the three goals below listing the information CLAMP needs from the user at each stage:

For seamless access to sources, need to know:

- what applications to merge

For cross fertilization of contexts, need to know:

- what are the isomodifiertypes, isomodifiers, isoattributes and isocontexts
- what semantic type, modifier, attribute and context names to “pull up” app
- what modifier values to assign to newly inherited modifiers for those contexts that do not have values for that modifier

For Extending Application:

- If adding context, then need to set modifier values for that context
- If adding source, need to elevate the columns of that source to semantic types
- If adding modifiers, need to enter a value for that modifier for each existing context
- If adding semantic type, need to specify inheritance
- If adding attribute, need to specify which semantic type it is an attribute of and what semantic type it leads to
- If adding conversion functions, need to enter which modifier it is for and need Prolog code of the conversion function

Given that the user will have to make various decisions in providing the information above, the question arises, how should CLAMP display the ontology and context information in order to best facilitate the user's merging decisions. The approach CLAMP adopts is to present relevant views of the application. That is, at each step, allow the user to see only that ontology information that is most helpful in making the particular decision at hand. For example, when deciding whether two semantic types should be declared isomodifiertypes (i.e. equivalent with respect to their modifiers), a user would be most helped by being able to see all the modifiers of those two semantic types. CLAMP does not provide much more hand-holding beyond that – no suggestions as to which semantic types should be isomodifiertypes, which semantic type may have implicit modifiers, etc.

The CLAMP approach may seem minimalist compared to other ontology merging tools and indeed it is. CLAMP does not want to give the user suggestions of what semantic types to equate, what contexts to reconcile, etc. This is because CLAMP's purpose is different than the

goal of traditional ontology merging. CLAMP is dealing with a much more specific, practical problem – merging existing data sources by merging the applications that provide access to those data sources. CLAMP is not interested in the more theoretical/general problem of materialized ontology merging where the desired result is an elegantly merged ontology with no concept redundantly represented by multiple semantic types. In materialized merging, all matching entities between the ontologies must be reconciled. This could potentially be a long list and so a list of suggestions is well-appreciated by the user. But in virtual merging, only a small subset of the ontologies needs to be reconciled – you do not have to resolve every case of matching semantic types but only those that will prevent proper context mediation. Thus a user can manage without a list.

Furthermore, CLAMP deals with ontologies that are already implemented in practical applications and so it assumes that the user is knowledgeable of those applications and knows what he wants. Suggestions of what merging decisions to make do not add much utility for a CLAMP user since he would need to know the applications thoroughly anyway in order to approve the suggestions. Meanwhile, the cost of giving good ontology/application merging suggestions is high because coming up with heuristics for good merging suggestions is a relatively difficult theoretical problem. Thus the minimalist approach: provide relevant application/ontology information to the user, let him make the decisions and let CLAMP focus on providing the important benefit of quick, error-resistant merger axiom generation without the need to know detailed Prolog.

4.4 - Design Overview

To present an overview of the design of CLAMP, we discuss two key aspects: its integration with Application Editor and the overall architecture. The following two sections cover these two aspects in detail.

4.4.1 – Integrated with Application Editor

CLAMP is designed to be integrated closely with Application Editor. There are two key reasons for this:

- 1) **Merged applications observe same ECOIN model as any other ECOIN application.**

As discussed earlier, merged applications are to be seen as any other standalone ECOIN application. They have ontology, context and sources just like any other application. The difference is only in their representation. Rather than their information being stored in one application prolog file, it is stored across multiple files (i.e. the files of the applications that were merged), which are linked by merger axiom files. But since the merged application is still one ECOIN application, it must be visible and editable through Application Editor. The internal ECOIN model data abstraction of Application Editor, the various Application Editor functions (such as

getSemanticTypes(), getAttributes()) are all relevant to merged applications and so it makes most sense that the CLAMP tool should be integrated with Application Editor and make use of the abstractions and functions it provides.

2) **CLAMP and Application Editor are both ECOIN application tools and so should have similar look-and-feel.**

If CLAMP is to be part of a suite of tools that allow one to develop, edit and maintain ECOIN applications, then it makes sense that these tools should have similar look-and-feel for the user's ease of understanding and use. Thus CLAMP is designed to have a similar look-and-feel as Application Editor.

4.4.2 – Architecture

The architecture of CLAMP is summarized in Figure 4.1 below. The dashed lines surround the pieces, which considered together, constitute CLAMP. Much of what is within the dashed lines is also a part of Application Editor. This is because CLAMP re-uses as much of the abstractions and code of Application Editor as it can and is designed to be embedded within Application Editor rather than be a stand-alone tool.

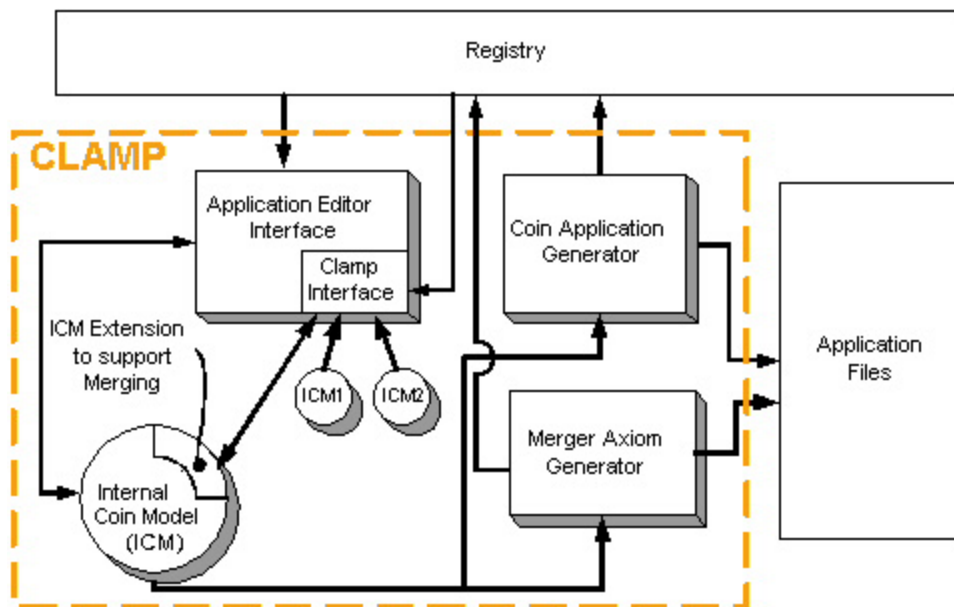


Figure 4.1 – Architecture of CLAMP

Looking at Figure 4.1, we begin with the internal COIN model (ICM). The ICM is a data abstraction that stores the COIN model metadata of an application during runtime. That is, it stores an application's ontology, context and source information. The ICM was originally created as part of the Application Editor architecture but it is also part of the CLAMP architecture because the COIN model abstraction is the same for a merged application and a standalone application. However, now that the ICM stores merged application metadata, the abstraction has to be expanded to include the new types of metadata that merging has

introduced (for example, information regarding isomodifiertypes, the underlying applications, etc). Thus the ICM in figure 4.1 contains a small subpart relating to merge information.

Next, we see the two ICMs directly beneath the CLAMP interface. These represent the COIN models of the two applications being merged. The CLAMP interface displays relevant COIN model information from these two applications to the user to assist him in his merging decisions. No changes are made to these two underlying applications during merging.

Next is the CLAMP interface. We have mentioned that the CLAMP interface displays relevant COIN model information from the applications being merged. It also prompts the user for isomodifiertype declarations, modifier values, etc. It then writes this information to the ICM of the merged application.

Next is the Application Editor interface. This interface is considered part of the CLAMP architecture because it is used for Goal 3 of merging – extending the merged application. Why use the Application Editor interface for this merging goal? Because the merged application is just another ECOIN application, hence the Application Editor interface should suffice to extend the application.

Next is the ECOIN Application generator. This generator takes information from the ICM and translates it into Prolog. It was developed as part of Application Editor and it takes the ICM through a series of XSLT transformations from RDF to RuleML to RFML to Prolog (see [12]). This generator is used in the CLAMP architecture for Goal 3 – extending the merged application.

Finally, there is the Merger Axiom generator. This generator takes the information from the ICM (chiefly the information stored in the merging subpart) and generates merger axioms.

The information from both the ECOIN application generator and the Merger Axiom generator is written to a merger axiom file, which becomes part of the set of application files representing the existing ECOIN applications. The generators also write the location of these application files to the registry so that they can be retrieved by the interfaces to give to the ICMs.

4.5 – CLAMP Interface – How the user is led through merging

As discussed earlier, CLAMP’s approach is to lead the user through the three goals of the merging process, giving the user relevant information at each stage to assist with each merging decision. Accordingly, the interface has pages that correspond to each goal of merging and a page that displays the merger axioms generated. To allow easy navigation, there are buttons for each of these pages that appear at the top of every page:

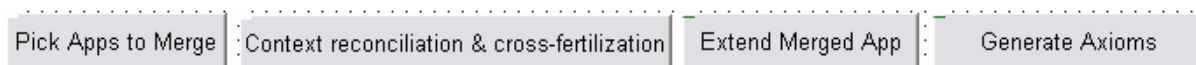


Figure 4.2 – CLAMP navigation buttons

The first three buttons correspond to the first three goals while the last button generates axioms.

4.5.1 – First page: Seamless Access to Sources

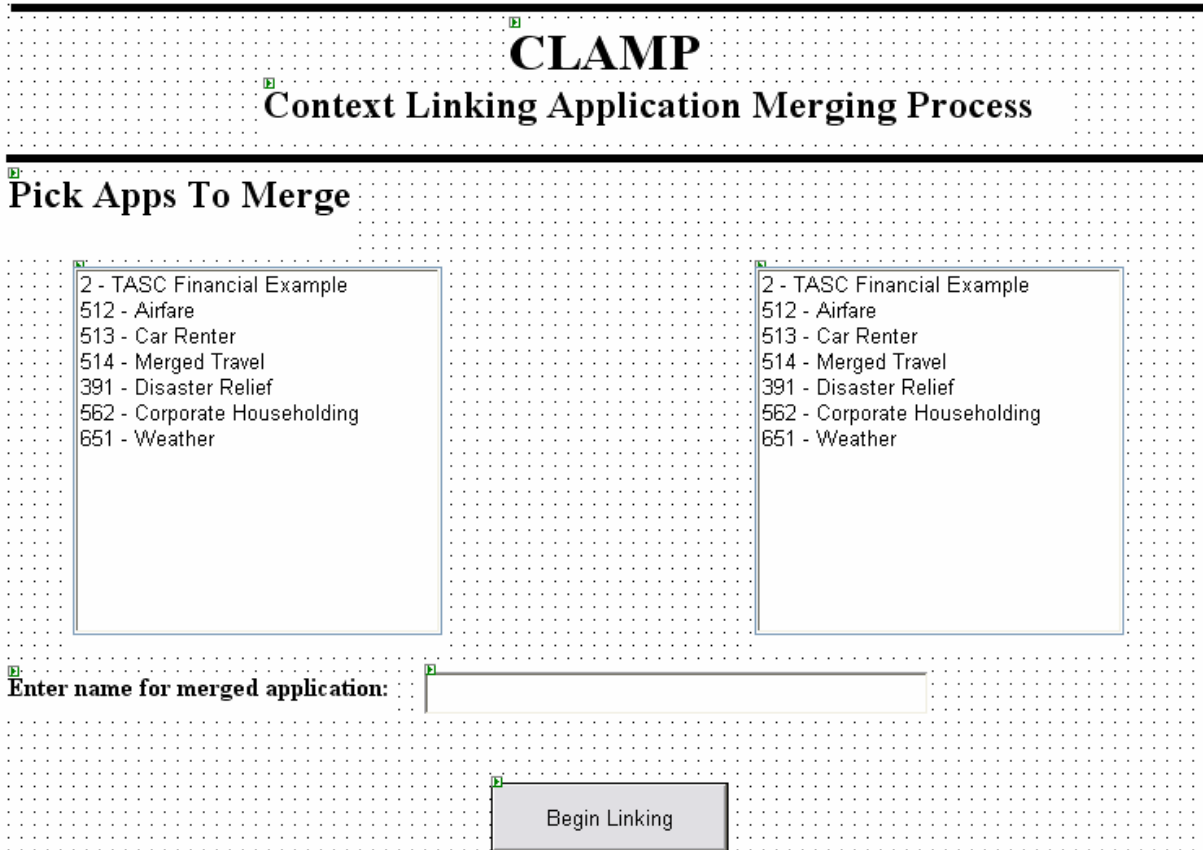


Figure 4.3 – CLAMP first page – pick two apps to provide access to sources from both

The first screen of the CLAMP interface presents the user with two lists, each of which contains all the active ECOIN applications in the registry. The user is prompted to pick two applications to merge and to enter a name for the merged application. When the user hits the “Begin Linking” button, the two applications picked are stored in the ICM (see section 4.4.2). The merger axiom generator uses this information to write the axiom that provides access to all sources from both applications (see Section 3.4 for details of axioms).

4.5.2 – Second page – context reconciliation and cross-fertilization

Context Reconciliation and Cross-Fertilization

Merging Applications 512 and 513 - Airfare and Car Rental

Semantic Types in App512 cityOrAirport moneyAmount price flight country	<input type="radio"/> is isomodifiertype of 1 <input checked="" type="checkbox"/> call it moneyAmount in merged app <input type="checkbox"/> call it moneyAmount2 in merged app <input type="checkbox"/> name it as follows in merged app: _____	Semantic Types in App513 rental city moneyAmount2 ratePeriod price2 date
Modifiers for SemType selected above currency	<input type="radio"/> is isomodifier of 2 <input type="checkbox"/> call it currency in merged app <input type="checkbox"/> call it [] in merged app <input type="checkbox"/> name it as follows in merged app: _____	Modifiers for SemType selected above
Attributes for SemType selected above	<input type="radio"/> is isoattribute of 3 <input type="checkbox"/> call it [] in merged app <input type="checkbox"/> call it [] in merged app <input type="checkbox"/> name it as follows in merged app: _____	Attributes for SemType selected above
<input type="button" value="Add 'Iso-' Relationships"/>		

Isocontexts

Contexts for App512 dora c_expedia c_yahoo c_itn c_orbitz c_travelselect doras_friend olsen_context	<input type="radio"/> is isocontext of 4 <input type="checkbox"/> call it [] in merged app <input type="checkbox"/> call it [] in merged app <input type="checkbox"/> name it as follows in merged app: _____	Contexts for App513 joe c_yahoocar c_orbitzcar c_qixocar joes_friend
<input type="button" value="Add Isocontext Relationship"/>		

"Iso-" Relationships so far

city is isomodifiertype of cityOrAirport and cityOrAirport is pulled up 5	<input type="button" value="Delete"/>
--	---------------------------------------

Figure 4.4 – Part of the Context reconciliation & cross-fertilization page

The second page of the interface is the most significant – the meat of the interface. It prompts the user through context reconciliation and context cross-fertilization. The page has three subsections. Figure 4.4 shows the first two subsections.

The main job of the first subsection (the areas labeled “1”, “2” and “3”) is to assist the user in declaring isomodifiertypes. To this end, CLAMP displays the semantic types of both applications (in area 1). When a semantic type is selected, its modifiers and attributes are automatically displayed (in areas 2 & 3) making it easier for the user to decide whether two semantic types can be considered isomodifiertypes i.e. equivalent with respect to their modifiers. When one semantic type from each application is selected, the checkboxes between the two lists give the user the option of choosing which semantic type name to “pull up” into the merged application (or whether to use a new name in the merged application). Furthermore, when the modifiers of the two semantic types are displayed (area 2), the user can denote isomodifiers and pick one of the modifier names to be pulled up (or give it a new name). The user can similarly denote isoattributes (area 3) and pull up the attribute name he chooses. Note that the user does not have to check off isomodifiers and isoattributes when setting isomodifiertypes because it is not necessary that there be any isomodifiers or isoattributes.

The screen shot above shows the example of merging Airfare and Car Rental. The semantic types `moneyAmount` and `moneyAmount2` are selected causing their modifiers and attributes to be automatically displayed - `currency` is the modifier of `moneyAmount` while `moneyAmount2` has no modifiers. Neither semantic type has any attributes. This is all reflected in the textboxes and the checkboxes in areas 2 and 3. The checkboxes in area 1 give the option of choosing either semantic type name to be pulled up into the merged application.

The next section on the page (area 4) is for denoting isocontexts. The contexts of each application are displayed in two textboxes and similar to all the other “iso” declarations thus far, the user can denote isocontexts and which context name to pull up (or rename).

The textbox in area 5 displays all the “iso” relationships as they are added and also allows the user to delete from that list. Each time the “Add iso relationship” or the “Add isocontext” button is clicked, the information is added to the internal COIN model (ICM) of the merged application that is being maintained during runtime. This ICM will be translated into Prolog when the Generate Axioms button is clicked.

There is one more section on this page in the interface (see figure 4.5). This section is for entering modifier values. Namely, declaring isomodifiertypes causes modifiers inherited from one underlying application to become relevant in contexts inherited from the other underlying application and thus modifier values need to be assigned for those contexts. For example, after declaring `moneyAmount` (from Airfare) and `moneyAmount2` (from Car Rental) as isomodifiertypes, the contexts from Car Rental now need to be assigned a value for the modifier `currency` that was inherited from Airfare. Figure 4.5 shows how this is done. All the unassigned modifiers are automatically displayed in one textbox and when a particular one is selected, CLAMP displays all the contexts for which that modifier has no value. The user

can then enter a modifier value (or how to derive a value if it is to be dynamic). Once the “Assign modifier value” button is clicked, it is entered into the ICM and shows up in the textbox showing all the modifier values thus far from which the user can delete as well.

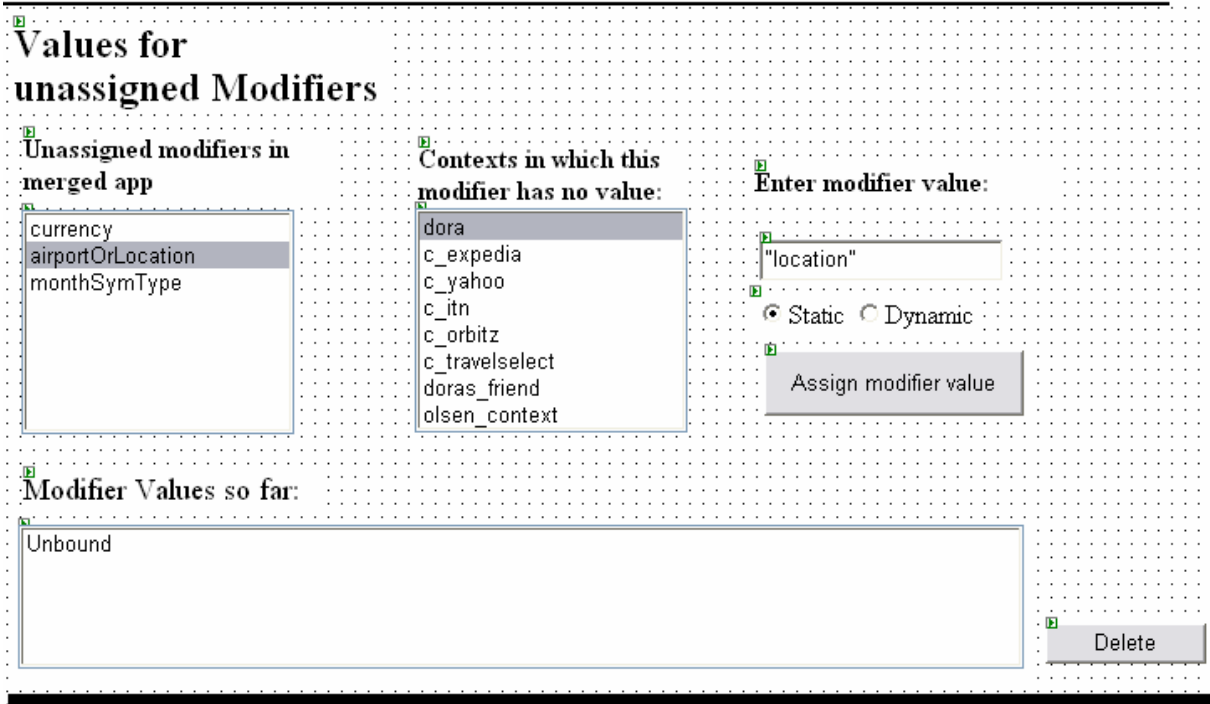


Figure 4.5 – Entering values for unassigned modifiers

4.5.3 – Extending the merged application

To achieve the third goal of merging, extending the merged application, the user is sent to the Application Editor because the merged application is considered just another ECOIN application. Thus adding semantic types, contexts, sources, etc, can all be done in the Application editor. Use of Application Editor’s interface is described in detail in [12].

4.5.4 – Generating merger axioms

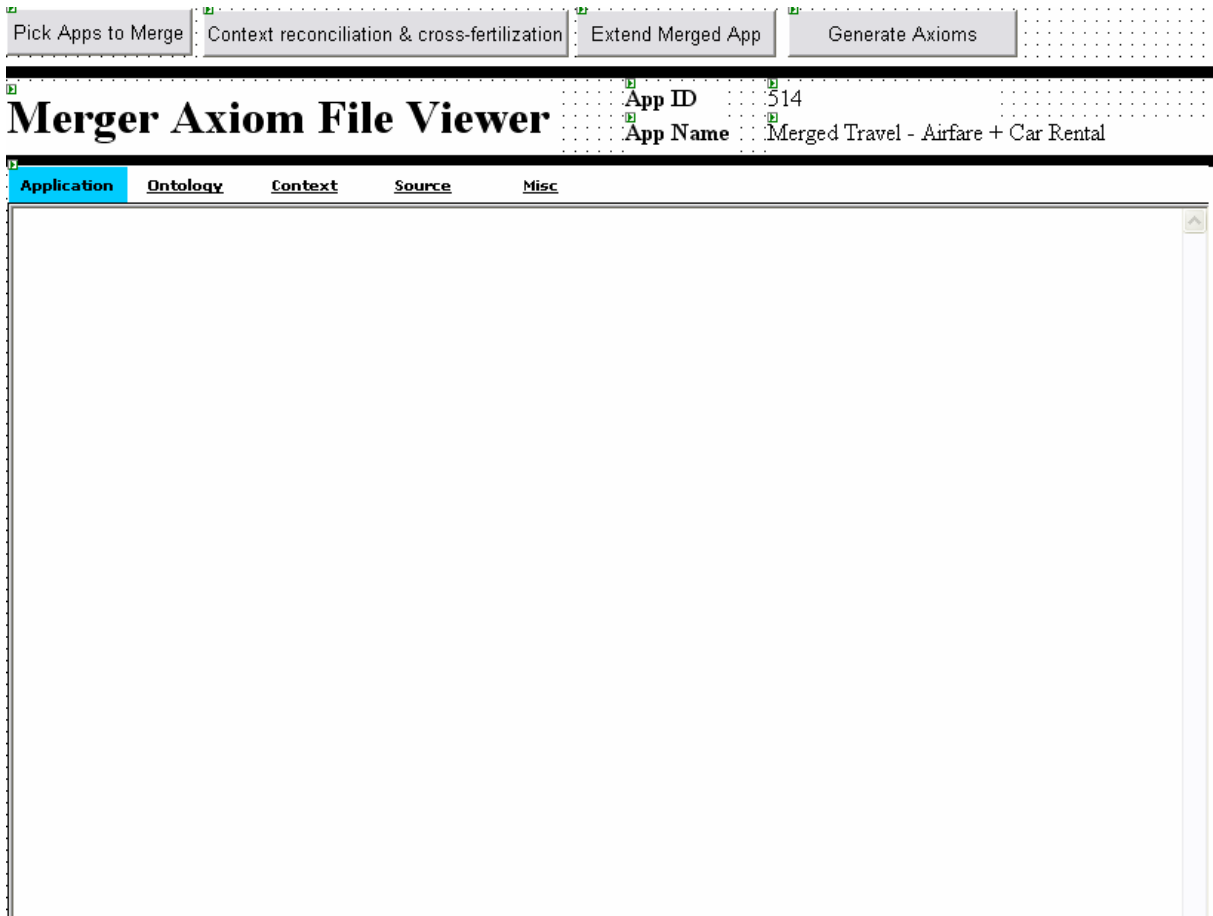


Figure 4.6 – Viewing Merger Axioms

When the user clicks on the Generate axioms button, he is brought to the page shown in Figure 4.6. The ICM axioms generated are displayed in the main textbox on the page.

The interface uses the Merger Axiom generator to create the axioms. The Merger Axiom generator is a separate program that takes an ICM data structure as an input and generates a file of merger axioms in Prolog as the output. The location of this file is stored in the Registry so that the interface can retrieve it later if needed.

The ICM contains all the information necessary to generate the axioms (section 4.6.1 discusses all the information that the ICM stores). The generator simply has to iterate through the data structure and convert the information into Prolog syntax.

4.6 – How Application Editor needs to be modified to support CLAMP

As discussed earlier, CLAMP is integrated closely with Application Editor. But for this to be possible, Application Editor needs to be modified – the implementation of the ICM and the API that allows access and change to the ICM need to be modified. The next two sections outline the needed modifications to the ICM and the API.

4.6.1 – How the Internal COIN Model (ICM) must be changed

As defined in [12], the Internal COIN Model (ICM) is a transient data structure that stores the application metadata (the application’s COIN model) at run time. The COIN model of an application is the ontology, context and source information. Merged applications are satisfactorily described by the current COIN model. However, an ICM that represents a merged application’s COIN model must be implemented differently because it needs to know about the two underlying applications, the isomodifiertypes, isomodifiers, isoattributes, and the isocontexts. In this section we describe how the ICM needs to be modified.

First, we present the ICM as described in [12]: The ICM is a data structure in object-oriented programming language. In object-oriented programming language, data structures are called *classes* and the functions that operate on these classes are *methods*. Classes have *properties*, i.e. the characteristics of the classes. Each property stores a single data type, whether it is a string, a number, or another class. Figure 4.7 shows a summary of the ICM as it is currently modeled. *Coin_Model*, *Ont_SemanticType*, *Ont_Attribute*, *Ont_Modifier*, *Cxt_Context*, *Src_Relation*, *Src_Column*, and *Src_ElevatedRelation* are class objects of the ICM. The bullet points under each of these classes are their properties. Figure 4.7 is reproduced directly from [12] where it is discussed in detail - thus we will not discuss it here. Rather, we discuss below the additions that need to be made to the ICM to support merging.

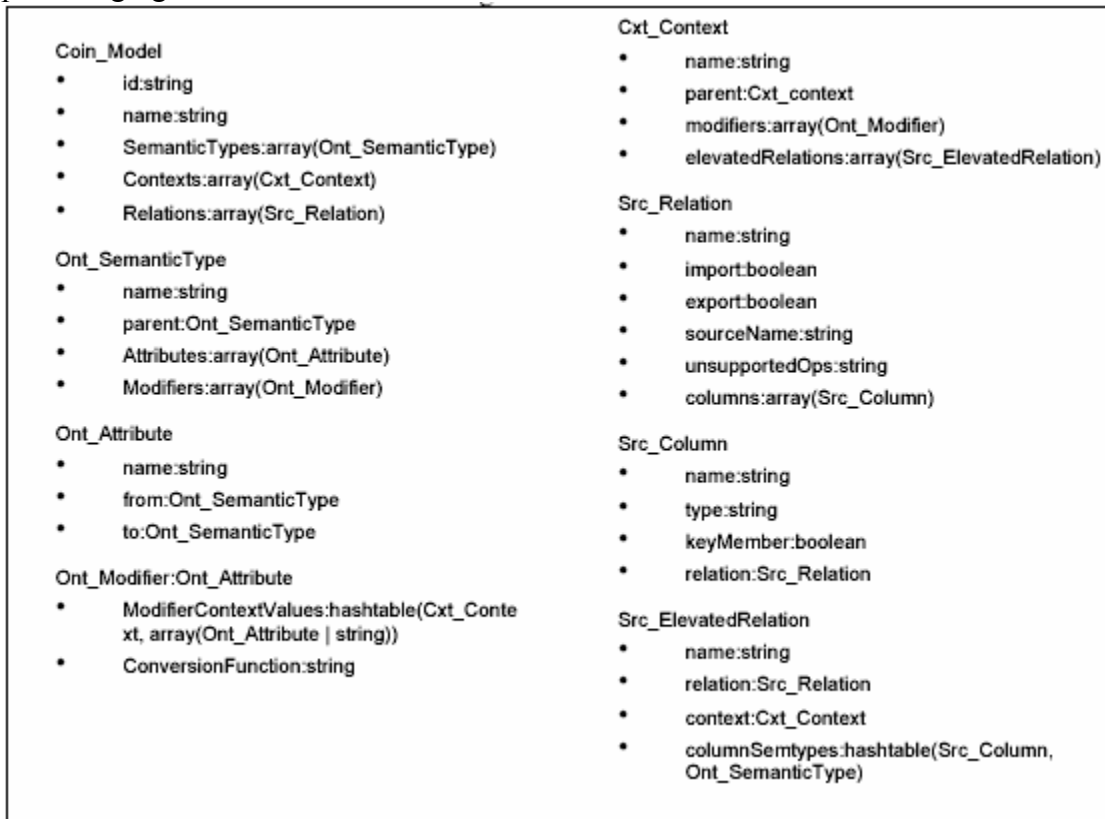


Figure 4.7 – Summary of the ICM as it exists in Application Editor

ICM Coin_Model

The `Coin_Model` class in ICM is the top-level data structure from which all the components of a COIN model can be reached. To support merging, `Coin_Model` should contain the following properties (property name in italics, property class type in courier font):

```
Id: string
Merges : array (Coin_Model)
Name: string
SemanticTypes: array (Ont_SemanticType)
Contexts: array (Cxt_Context)
Relations: array (Src_Relation)
Isomodifiertypes : array (Ont_Isomodifiertypes)
Isomodifiers : array (Ont_Isomodifiers)
Isoattributes : array (Ont_Isoattributes)
Isocontexts : array (Cxt_Isocontexts)
```

The properties in bold are the new ones added to support merging. We discuss only those because the rest are covered in detail in [12]. The *merges* property contains an array of the `Coin_Models` of the two underlying applications being merged. If this application has no underlying applications, then this property will be null. The *merges* property is sufficient to allow access to every aspect of the underlying applications because one can reach all ontology, context and source components from a `Coin_Model`.

The *Isomodifiers* property contains an array of `Ont_Isomodifiertypes` – a new class that is being added to the ICM to represent isomodifiertypes in the merged application (see `Ont_Isomodifiertype` section below). Similarly, the *Isomodifiers*, *Isoattributes* and *Isocontexts* properties are new to `Coin_Model` and they are all arrays of new classes that are being added to the ICM – `Ont_Isomodifier`, `Ont_Isoattribute` and `Cxt_Isocontext`.

The properties for the `Ont_Isomodifiertype` class are as follows:

Ont_Isomodifiertype

```
Name: string
SemanticTypes: array (Ont_semanticType)
FromApps: array (string)
```

The property, *Name* is a string that represents the name that is pulled up into the merged application. For example, if semantic types `price` and `cost` are declared isomodifiertypes, then either the name “price” or the name “cost” has to be pulled up into the merged application (or a new name has to be given). Whatever the case may be, this name is reflected in the *Name* property. The *SemanticTypes* property is an array of the two `Ont_semanticTypes` that have been declared isomodifiertypes. From this array, the ICM

can access all the modifiers, attributes, etc of the semantic types that have been declared `isomodifiertypes`. Finally, the `FromApps` property is an array of two `strings` – the application ids of the applications from which the two semantic types are drawn. The first id in the array corresponds to the first semantic type in the `Ont_SemanticTypes` array and the second id corresponds to the second `Ont_SemanticType`.

The properties for the remaining three new classes, `Ont_Isomodifier`, `Ont_Isoattribute` and `Cxt_Isocontext` are listed below. They are not described in detail because they function in a manner analogous to `Ont_Isomodifiertype`:

Ont_Isomodifier

Name: `string`

Modifiers: `array(Ont_modifier)`

FromApps: `array(string)`

Ont_Isoattribute

Name: `string`

Attributes: `array(Ont_attributes)`

FromApps: `array(string)`

Cxt_Isocontext

Name: `string`

Contexts: `array(Cxt_context)`

FromApps: `array(string)`

Despite the changes to the ICM described above, the Application Programming Interface (API) that is used to communicate with the ICM will still have the same functions with the same signatures (i.e same input parameters and same output types) as developed in [12]. However, the implementation of many of these functions will have to be changed to support merging. These changes are discussed in the next section. Furthermore, a few more API functions will need to be added to interact with the new classes and properties that were added to the ICM. We discuss these additions and modifications to the API in the next section.

4.6.2 – How the API must be changed

Merged applications maintain all the abstractions of an ECOIN application but the implementation has changed in one key way. The ontology, context and source information is now potentially stored across multiple Prolog files. Namely, a set of merger axioms points to two application files, one of which may be another set of merger axioms and so on. Meanwhile, Application Editor assumes that application information is stored in one Prolog file and so all the functions in its API are implemented to read and make changes to only one application file. To support CLAMP, the API's implementation has to be changed such that it knows to deal with multiple application files for a single ECOIN application.

Specifically, a function must first determine whether the application that is currently loaded is a merged application. If so, it needs to determine the underlying applications (and continue down the chain if the underlying applications are also merged applications) and load the ontology, context and source information from all the constituent applications into the ICM that is maintained during runtime. Note, however, that the API cannot just blindly load all the ontology, context and source information from the underlying applications but instead must make sure that it does not load application information that has been overridden by any merger axioms along the way. Furthermore, when a merged application is edited or extended, the API must make sure to write these changes only into the merger axioms file and not into the underlying applications' files (because those are still used by existing applications).

Rather than making all the changes described in the preceding paragraph to every API function that needs it, the changes should be encapsulated within a couple of internal utility functions. These functions should then be used by the API functions that need to be modified.

Besides the above changes to the API, new functions also need to be added to support changes to the ICM due to merging (see Section 4.6.1). Namely, functions needed to be added that will allow the CLAMP interface to set isomodifiertypes, isoattributes, isomodifiers and isocontexts within the ICM of an application. Note that no functions need to be added to remove an isomodifiertype or isoattribute, etc. This is because a merged application is like any other ECOIN application. So an isomodifiertype link is removed by removing the semantic type name that was pulled up to represent that isomodifiertype. For example, if `price` and `moneyAmount` were set as isomodifiertypes, and `price` was pulled up, then the isomodifiertype relationship between `price` and `moneyAmount` would be removed by simply removing “price” from the merged application. The implementation of the existing `removeSemanticType()` function in Application Editor will have to be modified so that the function knows how to determine whether this is an isomodifier involved and take care of any related cleanup. The same applies to the `removeAttribute()`, `removeModifier()` and `removeContext()` functions from Application Editor.

Thus only four new functions need to be added to the API:

```
public void createIsoModifierType ()
public void createIsoModifier ()
public void createIsoAttribute ()
public void createIsoContext ()
```

The details of these functions are in Appendix D and the full ICM API can be found in [12].

5. Possible improvements/extensions

After undertaking the detailed case study of context-linking and designing the CLAMP tool, there are a few possible improvements/extensions that can be pursued in the future.

5.1 – Graphical Enhancement

The context-linking process is a very ontology-centered process. That is, the developer is making decisions regarding which semantic types are equivalent, which modifiers perform the same function, etc. This is all information that is captured graphically in the ontology diagram and so the developer would benefit a lot if he were able to see the two ontologies in detail, be able to manipulate these ontologies and even denote the context links graphically. Currently, this is not possible – what happens instead is that the developer is provided relevant ontology information in text format and if he likes, he can look at pictures of the two ontologies separately. However, we suggest this improvement with caution, developing good graphical tools to manipulate ontologies is a difficult problem because appropriate screen placement of the various ontology objects is difficult. Application Editor provides a limited graphical interface that draws ontologies of existing applications but does not allow changes to the ontology (see [12])

5.2 – Implicit modifier detection

During context reconciliation, the developer has to carefully think about each of the semantic types in the application to determine whether any implicit assumptions are being made regarding how that semantic type is to be interpreted. For example, in an application that deals solely with US data sources, an assumption might be made that all currency is in US dollars. This modifier would have to be made explicit when being merged with an international application. Currently, no assistance is provided to the developer to detect implicit modifiers and so CLAMP stands to benefit from such an addition. It is not easy to design an automated way to detect implicit modifiers but rather than solving the problem completely perhaps the developer can be assisted. Each time a semantic type is clicked, CLAMP can automatically list all the data source columns that have been elevated to that semantic type. Thus the user will be able to tell more easily whether any assumptions are being made about that semantic type.

5.3 – Conversion function assistant

In merging two applications, modifiers that originated in one application are assigned values in contexts that originated in the other application, new modifier values might be created, etc. Thus it is quite probable that new conversion functions will have to be written. One of the goals of CLAMP was to save the developer from having to deal with Prolog. However, currently, there is no good way to create conversion functions without actually writing them in Prolog. Application Editor provides a library of common functions that the user can select

from but most functions would still have to be written by hand. As more and more ECOIN applications are created, however, the library of conversion functions will grow and even if it is still difficult to find a conversion function that matches what a developer is looking for, he may at least find several similar functions, which he can modify or imitate and avoid writing all the Prolog himself. However, this is a roundabout manner of development and both CLAMP and Application Editor stand to benefit from the development of a conversion function assistant.

6 - Related Work

There exist various research efforts regarding automated or tool-supported merging of ontologies (sometimes called object-oriented schemas, database schemas, class hierarchies, domain models, or object models, depending on the field). In this section, we survey some of the more prominent research efforts/tools related to ontology merging and note the differences between those approaches and the context-linking approach/CLAMP tool.

Automated and user-supported ontology merging tools essentially focus on what we have termed materialized merging – that is merging two existing ontologies to yield a new merged ontology. The usual approach is to detect various classes of conflicts and suggest solutions for these conflicts. The methodology of detecting conflicts and suggesting solutions, the sophistication of the suggestions, and the types of ontology conflicts that the tools can detect and resolve is what distinguishes the different efforts from each other.

The approach of many of the tools is to make suggestions to the user as to which parts of the ontology he should consider merging. Currently, these suggestions focus mainly on finding syntactic matches between the two ontologies [14]. For example, if one ontology has *price* and another has *cost*, the tool should identify these two as candidates for a merge. To find such syntactic matches (and ones that are more subtle), there are various possible methods that merging tools utilize [4]. Some of the more common methods are:

- 1) looking for entities in the ontologies with similar or same names
- 2) relying on various dictionary and thesaurus systems to find synonyms for entities in an ontology and then checking whether those synonyms exist in the other ontology
- 3) evaluating common substrings in the names of the entities in the ontologies
- 4) looking to see if the documentation of entities in the ontologies share many uncommon words

Indeed some of these synonym-finding methods are topics of research in their own right. One such research effort, Wordnet [13], by the Cognitive Science Laboratory at Princeton University. WordNet is an on-line lexical reference system where English nouns, verbs, and adjectives are organized into synonym sets, each representing one underlying lexical concept. Different relations link the synonym sets. [13]. Any word that is typed into Wordnet will first yield the various “senses” of the word – i.e. the various semantic understandings that exist for that word. A user can then pick one of these “senses” and look up the synonyms of that sense. Going further, the user can even look up the “hypernyms,” “hyponyms,” “meronyms” (i.e. “superclass,” “subclass” and “attributes”), and the synonyms of each of those. For example, suppose we look up “boy”. Wordnet yields the following result:

The **noun** "boy" has 4 senses in WordNet.

1. male child, **boy** -- (a youthful male person; "the baby was a boy"; "she made the boy brush his teeth every night"; "most soldiers are only boys in uniform")
2. **boy** -- (a friendly informal reference to a grown man; "he likes to play golf with the boys")

3. son, **boy** -- (a male human offspring; "their son became a famous judge"; "his boy is taller than he is")
4. **boy** -- (offensive term for Black man; "get out of my way boy")

For any one of these senses, the user then has the following options:

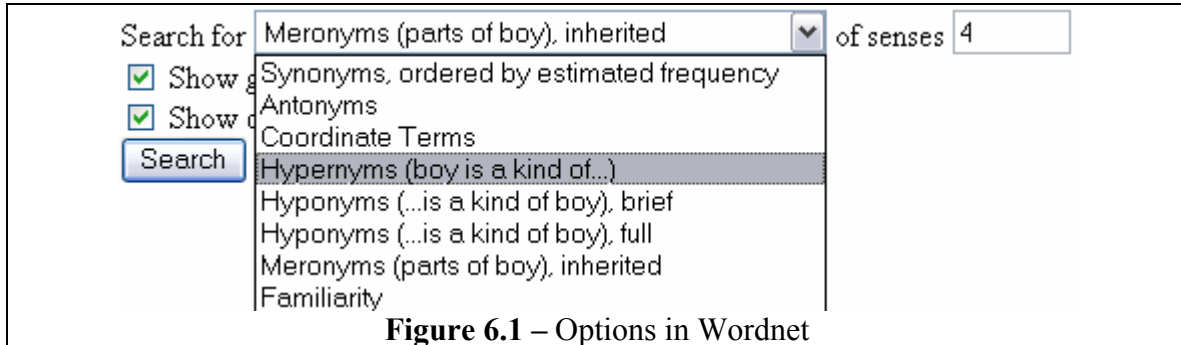


Figure 6.1 – Options in Wordnet

Of all the options in the dropdown box of Figure 6.1, synonyms, hypernyms, hyponyms and meronyms are the most relevant to merging. If we explore sense 3 (“boy” as in “son”), we net the following results (Wordnet results also include definitions of each term but some of these have been removed from the results below for the sake of brevity):

Synonyms:

Sense 3

son, boy -- (a male human offspring; "their son became a famous judge"; "his boy is taller than he is")
 => male offspring, man-child -- (a child who is male)

Hypernyms (boy is a kind of...):

Sense 3

son, boy -- (a male human offspring; "their son became a famous judge"; "his boy is taller than he is")
 => male offspring, man-child -- (a child who is male)
 => child, kid
 => offspring, progeny, issue --
 => relative, relation --
 => person, individual, someone, somebody, mortal, human, soul --
 => organism, being --
 => living thing, animate thing -- (a living (or once living) entity)
 => object, physical object --
 => entity, physical thing --

Hyponyms:

Sense 3

son, boy -- (a male human offspring; "their son became a famous judge"; "his boy is taller than he is")
 => Junior, Jr, Jnr -- (a son who has the same first name as his father)
 => mother's boy, mamma's boy, mama's boy
 (a boy excessively attached to his mother; lacking normal masculine interests)
 => Esau
 (Old testament) the eldest son of Isaac who would have inherited the covenant that God made with Abraham and that Abraham passed on to Isaac; he traded his birthright to his twin brother Jacob for a mess of pottage)

Meronyms:

Sense 3

son, boy -- (a male human offspring; "their son became a famous judge"; "his boy is taller than he is")

=> male offspring, man-child -- (a child who is male)

=> child, kid --

=> offspring, progeny, issue --

=> relative, relation --

=> person, individual, someone, somebody, mortal, human, soul --

HAS PART: personality

HAS PART: trait -- (a distinguishing feature of your personal nature)

HAS PART: character, fiber, fibre --

HAS PART: thoughtfulness -- (the trait of thinking carefully before acting)

HAS PART: responsibility, responsibility --

HAS PART: integrity -- (moral soundness)

HAS PART: nature --

HAS PART: human body, physical body, material body, soma, build, figure, physique, anatomy, shape, bod, chassis, frame, form, flesh --

=> organism, being --

HAS SUBSTANCE: tissue --

HAS PART: cell --

Overall, Wordnet is a powerful tool that can be leveraged by ontology merging tools to give suggestions regarding not just synonyms, but also suggestions regarding inheritance and attributes.

An example of a prominent tool that gives suggestions during merging is PROMPT [14]. PROMPT takes two ontologies as input and guides the user in the creation of one merged ontology as output by means of the following general algorithm [14]:

Creates an initial list of matches based on class names (equivalent to semantic types). Then go through the following cycle:

- a. User triggers an operation by either selecting one of the suggestions or by using an ontology-editing environment to specify the desired operation directly
- b. PROMPT performs the operation, *automatically* executes additional changes based on the type of the operation, generates a list of *suggestions* for the user based on the structure of the ontology around the arguments to the last operation, and determines *conflicts* that the last operation introduced in the ontology and finds *possible solutions* for those conflicts. These *suggestions* and *possible solutions* are presented to the user again and the cycle is repeated.

For example, suppose a user is merging two ontologies and he accepts one of PROMPT's initial suggestions to merge two classes (the PROMPT equivalent of semantic types) A and B to create a new class M. PROMPT would then perform several actions automatically and put together a new list of suggestions. The following is an example of some of the automatic actions and suggestions [14]:

- For each attribute S that was attached to A and B in the original ontologies, attach the attribute to M. If S did not exist in the merged ontology, create S.

- For each superclass of *A* and *B* that has been previously copied into the merged ontology, make that copy a superclass of *M* (thus restoring the original relation). Do the same for subclasses.
- For each class (semantic type) *C* in the original ontologies to which *A* and *B* referred (either through an inheritance or attribute relationship), if *C* has not been copied to the merged ontology, suggest that it is copied to the merged ontology.

The user would pick one of these suggestions and the cycle would be triggered again. The key contribution that PROMPT's developers emphasize is that they are not just making suggestions based on syntactic matches but that they base most of their suggestions on the internal structure of the ontology, i.e. the relationship between the various semantic types, attributes, etc. [14] Furthermore, PROMPT researchers' evaluations yielded results showing that human experts followed 90% of the suggestions that PROMPT generated and that 74% of the total knowledge-base operations invoked by the user were suggested by PROMPT [14].

Another prominent tool is Ontomorph [3]. Ontomorph presents an initial list of matches to the user (based on pattern matching) and also defines a set of operations that can be applied to the ontologies (without making a specific suggestion as to which of these operations the user should perform). The user then looks at the initial set of matches and defines a set of operations that should be performed and Ontomorph applies those operations (thus permitting aggregate operations in a one step). However, there is no assistance to the user beyond the initial set of matches. The more impressive contribution of Ontomorph is that it provides a powerful rule language [3] to represent complex syntactic operations.

Compared to these tools, the context-linking approach/CLAMP tool is unique. Firstly, context-linking is a type of virtual merging – it does not try to actually merge two ontologies completely but simply creates a ECOIN application that *appears* to be relying on one merged ontology. This greatly simplifies the problem because only a small subset of the two ontologies needs to be reconciled rather than resolving every case of matching semantic types.

Secondly, the CLAMP tool does not “detect” any syntactic matches nor does it offer any suggestions to the user. Instead it provides relevant information to the user to assist him in making merging decisions and then implements his merging decisions in Prolog. However, CLAMP could readily be extended to provide suggestions using Wordnet; Cameleon [7] technology could easily be used to query Wordnet regarding semantic types or any other objects in the ontologies, and the results can be extracted by Cameleon and made available for CLAMP to then create suggestions for the user. However, this route was not chosen – see section 4.3 for a detailed discussion of the reasoning behind not providing any suggestions.

Finally, we take a look at ONIONS (*ONtological Integration Of Naive Sources*) – a methodology for conceptual analysis and ontological integration [15]. Unlike the above tools that rely on expert users with knowledge of the ontologies, ONIONS uses a hierarchical ontology library containing various generic ontologies that have been classified and merged. To make merging decisions, ONIONS matches the ontologies to be merged to those existing in the library. Very briefly, ontology integration in ONIONS is carried out as follows [8]:

- All concepts, relations, templates, rules, and axioms from a source ontology are represented in the ONIONS formalism
- Ontologies are integrated by means of a set of generic ontologies. This is the most characteristic activity in ONIONS, which can be briefly described as follows:
 - For any set of sibling concepts, the conceptual difference between each of them is inferred, and such difference is resolved by axioms that reuse - if available - the relations and concepts already in the library. If no concept is available to represent the difference, new concepts are added to the library.
 - When stating new concepts, the links necessary to maintain the consistency with the existing concepts are created. If conflicts arise with existing theories, a more general theory is searched which is more comprehensive. If this is impracticable, an alternative theory is created.

The source ontologies are explicitly mapped to the integrated ontology. Even if the source ontologies do not completely fit into the integrated ontology in the library, partial interoperability suffices.

Overall, ONIONS has a powerful and interesting approach which is instructive in thinking about ontology merging but limited since it can only work with ontologies from domains that are represented in the library system. (This is fine with the developers of ONIONS because they are concerned only with medical ontologies.)

7. Conclusions

This thesis has covered much over the last seventy pages and so, as a summary, we list below the key topics and contributions of this thesis. Next, we evaluate the research based on the original motivations presented in Section 1.4.1 and we end with a look to the future.

7.1 Summary of thesis and contributions

- 1) A high-level description of the problem of context differences when integrating heterogeneous data sources
- 2) A high-level description of the COIN approach to resolving that problem
- 3) A detailed description of two applications that implement the COIN approach (Airfare and Car Rental)
- 4) An introduction to the problem of ontology merging and two approaches to it: materialized versus virtual merging.
- 5) Context linking – virtual ontology merging specific to ECOIN applications
- 6) A detailed exposition of the context-linking approach to ECOIN application merging. This included a detailed case study of merging Airfare and Car Rental that demonstrated the viability of context-linking
- 7) Design for a tool (CLAMP) that facilitates ECOIN application merging, reduces merging time and relieves the user from the need to know Prolog

Of these topics, the central two were the presentation of ECOIN application merging through context-linking and the proposal for a tool that facilitates ECOIN application merging. Indeed, at the beginning of this thesis, we motivated ourselves to pursue these two topics with four reasons. We now evaluate how those motivations have been satisfied:

7.2 Evaluation of motivations for thesis

In Section 1.4.2, we motivated the work of this thesis with four key reasons. We revisit these reasons below and evaluate our research based on these motivations.

Value to be gained by merging ECOIN applications

An application may find it very beneficial to have access to the set of sources that are part of another application. Or an application may be able to leverage the context capabilities of another application. By merging Airfare and Car Rental, we have shown the benefits from merging two realistic applications – we developed a general travel application with airfare finding and car renting abilities and context capabilities from both applications (currency conversion, airport code conversion). Furthermore, we were able to extend the merged application and provide the ability to find a combined airfare and car rental price (the rental automatically being for the duration of the stay) simply by entering the destination and dates of travel. We have shown that these benefits can potentially be gained by merging any two applications (the “three goals” of merging). We have shown that merging is a much faster way to create a ECOIN application that covers two domains than developing such an

application “from scratch”. We have shown that the CLAMP tool cuts the time further by speeding up the context linking process. This time gains are significant when multiple applications are to be merged (done two at a time).

Re-use

With numerous applications and ontologies being developed independently, there is a lot of benefit to be derived from re-using work that has already been done. In this thesis, we have shown, through the example of merging Airfare and Car Rental, the re-use of sources from both applications and the re-use of portions of the ontology (which leads to inheriting context capabilities from both underlying applications). However, “re-use” has also meant “repeat.” That is, in re-using ontology, we did not show two ontologies being merged to yield an elegant, merged result – rather the virtually merged ontology contains many redundancies i.e. the same concept being captured by more than one semantic type.

Better fit with how ontologies/applications are developed in real life

In the real world, designers of applications (and ontologies) rarely have a broad enough vision to predict what will be desired in the future - ontologies, applications, and standards are constantly being developed and evolved by countless independent parties. It is better to adapt to this reality and design small, relevant applications and merge them with other applications as needed than to try to predict and pre-plan large, comprehensive applications.

We have shown that context-linking allows you to go about merging whatever two applications you want fairly easily as long as you see some benefit to be gained either – whether the benefit is as simple as gaining a new source or significant as gaining new context capabilities. Upward inheritance is what prevents all these applications being merged from becoming very unwieldy, Namely, all the underlying applications’ metadata is inherited implicitly into the merged application’s file and is not explicitly listed. If you were to draw out the resulting “virtually merged ontology” it would be huge and inelegant perhaps, but in Prolog implementation, it is very efficiently represented.

Easier problem to solve than “materially” merging ontologies

The general theoretical problem of ontology merging is a difficult artificial intelligence problem. In the related work section, we outlined various groups’ approaches to this problem and saw the difficulties they faced. On the other hand, we have shown that virtual merging through context-linking is a workable solution for merging ECOIN application ontologies. Rather than traditionally merging ECOIN ontologies and then developing an application around the merged ontology, ECOIN is content with quickly arriving at a virtually-merged ontology because it is provided the functionality it wants – context-mediation across the sources of both applications. Furthermore, the two underlying ontologies persist within the ECOIN system and are still usable by their original applications.

7.3 – A look ahead

Overall, we have shown that ECOIN Application merging through context-linking provides much benefit to the ECOIN system and satisfactorily addresses the factors that motivated the approach in the first place. One of the more interesting characteristics of ECOIN application merging is that it lends itself to unplanned, decentralized evolution. Namely, merging has been made easier, it does not affect existing applications, and much benefit is to be gained even from merging applications that are not obviously complementary. Thus, various independent developers can take initiative and merge various applications with other merged applications and so on. In general, such decentralized endeavors, for example, the Internet, evolve in directions and yield unpredicted future benefits that the original players did not even imagine. Thus as context-linking becomes more established and more and more ECOIN applications are merged, it will be very interesting to see what shapes the various unplanned webs of ECOIN applications take and what surprising results are attained.

References

- [1] Ayenew, Befekadu. Alignment and Merging for Collaborative Domain Spaces. Master of Engineering thesis proposal. Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Dec 2001.
- [2] Bressan, Stephane, Cheng Goh, Natalia Levina, Stuart Madnick, Ahmed Shah, and Michael Siegel. Context Knowledge Representation and Reasoning in the Context Interchange System. *Applied Intelligence* 13, 165-180, 2000.
- [3] Chalupsky, Hans. OntoMorph: A Translation System for Symbolic Knowledge. *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning*, San Francisco, CA, 2000.
- [4] Chalupsky, H., Hovy, E. and Russ, T. Progress on an Automatic Ontology Alignment Methodology. 1997
- [5] The Context Interchange Group Home Page,
http://context2.mit.edu/coin/description/database_world.html
- [6] Firat, Aykut. Information Integration using Contextual Knowledge and Ontology Merging. Ph.D. Thesis. Massachusetts Institute of Technology, 2003.
- [7] Firat, Aykut, Stuart Madnick and Michael Siegel. The Cameleon Approach to the Interoperability of Web Sources and Traditional Relational DataBases. *Proceedings of the 10th Annual Workshop On Information Technologies and Systems*, Brisbane, Queensland, Australia, 2000.
- [8] Gangemi, Aldo, Domenico M. Pisanelli and Geri Steve. An Overview of the ONIONS Project: Applying Ontologies to the Integration of Medical Terminologies. 1999
- [9] Goh, Cheng Hian. Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Systems. PhD dissertation, Massachusetts Institute of Technology, Sloan School of Management, December, 1996.
- [10] Goh, Cheng Hian, Stephane Bressan, Stuart Madnick, and Michael Siegel. Context Interchange: New Features and Formalisms for the Intelligent Integration of Information. *ACM Transactions on Information Systems*, Vol. 17, No. 3, July 1999, Pages 270-293.

- [11] Gruber, Thomas R. A Translation Approach to Portable Ontology Specifications. Knowledge Systems Laboratory, Stanford University, Technical Report KSL 92-71, April 1993
- [12] Lee, Philip. Metadata Representation and Management for Context Mediation. MIT Master of Engineering Thesis. Massachusetts Institute of Technology, 2003.
- [13] Miller, George A, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine Miller. Introduction to WordNet: An On-line Lexical Database. Cognitive Science Laboratory, Princeton University, 1993
- [14] Noy, Natalya F. and Mark A. Musen, "PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment." In *Proceedings of AAAI-2000*, Austin, Texas. MIT Press/AAAI Press, 2000, 450-455.
- [15] Ying Ding, Dieter Fensel, Michel Klein, Borys Omelayenko, "Ontology Management: Survey, Requirement and Directions." *On-to-Knowledge Project*. Division of Mathematics & Computer Science, Vrije Universiteit Amsterdam. 1999.

Appendix A – Application Prolog file from Airfare

```
:- module_interface(application512).
:- export rule/2.
:- begin_module(application512).
:- dynamic rule/2.

%%% domain model for Airfare Aggregator
%%% generation timestamp: 4/8/2003 10:06:23 PM

%%%
%%% Semantic types
%%%
rule(is_a(airline, basic), (true)).
rule(is_a(cityORAirport, basic), (true)).
rule(is_a(country, basic), (true)).
rule(is_a(currency, basic), (true)).
rule(is_a(date, basic), (true)).
rule(is_a(dateType, basic), (true)).
rule(is_a(day, basic), (true)).
rule(is_a(duration, basic), (true)).
rule(is_a(durationType, basic), (true)).
rule(is_a(exchangeRate, basic), (true)).
rule(is_a(flight, basic), (true)).
rule(is_a(includesServFee, basic), (true)).
rule(is_a(includesVisaFee, basic), (true)).
rule(is_a(includesPaperCharge, basic), (true)).
rule(is_a(moneyAmount, basic), (true)).
rule(is_a(month, basic), (true)).
rule(is_a(onTimeProbability, basic), (true)).
rule(is_a(paperTicketFee, moneyAmount), (true)).
rule(is_a(price, moneyAmount), (true)).
rule(is_a(provider, basic), (true)).
rule(is_a(serviceFee, moneyAmount), (true)).
rule(is_a(timeZone, basic), (true)).
rule(is_a(traveler, basic), (true)).
rule(is_a(trip, basic), (true)).
rule(is_a(visaFees, moneyAmount), (true)).
rule(is_a(year, basic), (true)).
rule(is_a(c_yahoo, basic), (true)).
rule(is_a(c_expedia, basic), (true)).
rule(is_a(c_orbitz, basic), (true)).
rule(is_a(c_itn, basic), (true)).
rule(is_a(c_travelselect, basic), (true)).
rule(is_a(olsen_context, basic), (true)).
rule(is_a(dora, basic), (true)).
rule(is_a(doras_friend, basic), (true)).
rule(is_a(c_demo, basic), (true)).

rule(contexts([c_yahoo, c_expedia, c_orbitz, c_itn, c_travelselect, olsen_context, dora, doras_friend, c_demo]),(true)).

%%%
%%% Modifiers
%%%
rule(modifiers(airline, []), (true)).
rule(modifiers(basic, []), (true)).
rule(modifiers(cityORAirport, []), (true)).
rule(modifiers(country, []), (true)).
rule(modifiers(currency, []), (true)).
rule(modifiers(date, [dateType]), (true)).
```

```

rule(modifiers(dateType, []), (true)).
rule(modifiers(day, []), (true)).
rule(modifiers(duration, [durationType]), (true)).
rule(modifiers(durationType, []), (true)).
rule(modifiers(exchangeRate, []), (true)).
rule(modifiers(flight, []), (true)).
rule(modifiers(moneyAmount, [currency]), (true)).
rule(modifiers(month, []), (true)).
rule(modifiers(onTimeProbability, []), (true)).
rule(modifiers(paperTicketFee, []), (true)).
rule(modifiers(price, [includesServFee, includesVisaFee, includesPaperCharge]), (true)).
rule(modifiers(includesServFee, []), (true)).
rule(modifiers(includesVisaFee, []), (true)).
rule(modifiers(includesPaperCharge, []), (true)).
rule(modifiers(provider, []), (true)).
rule(modifiers(serviceFee, []), (true)).
rule(modifiers(timeZone, []), (true)).
rule(modifiers(traveler, []), (true)).
rule(modifiers(trip, []), (true)).
rule(modifiers(visaFees, []), (true)).
rule(modifiers(year, []), (true)).

%%
%% Attributes
%%
rule(attributes(airline, []), (true)).
rule(attributes(basic, []), (true)).
rule(attributes(cityORAirport, [isIn, timeZone]), (true)).
rule(attributes(country, [visaFees]), (true)).
rule(attributes(currency, []), (true)).
rule(attributes(date, []), (true)).
rule(attributes(dateType, []), (true)).
rule(attributes(day, []), (true)).
rule(attributes(duration, []), (true)).
rule(attributes(durationType, []), (true)).
rule(attributes(exchangeRate, [txnDate, fromCur, toCur]), (true)).
rule(attributes(flight, [airline, duration, stopOver, timeliness]), (true)).
rule(attributes(moneyAmount, []), (true)).
rule(attributes(month, []), (true)).
rule(attributes(onTimeProbability, []), (true)).
rule(attributes(paperTicketFee, []), (true)).
rule(attributes(price, [destCntry, provider]), (true)).
rule(attributes(includesServFee, []), (true)).
rule(attributes(includesVisaFee, []), (true)).
rule(attributes(includesPaperCharge, []), (true)).
rule(attributes(provider, [paperTicketFee, serviceFee]), (true)).
rule(attributes(serviceFee, []), (true)).
rule(attributes(timeZone, []), (true)).
rule(attributes(traveler, [citizenship]), (true)).
rule(attributes(trip, [departureDate, departureDay, departureMonth, departureYear, destination, flight, origin, price,
returnDate, returnDay, returnMonth, returnYear, traveler]), (true)).
rule(attributes(visaFees, []), (true)).
rule(attributes(year, []), (true)).

%%
%% Contexts
%%
rule(context(c_yahoo), (true)).
rule(context(c_expedia), (true)).
rule(context(c_orbitz), (true)).
rule(context(c_itn), (true)).

```

```

rule(context(c_travelselect), (true)).
rule(context(olsen_context), (true)).
rule(context(dora), (true)).
rule(context(doras_friend), (true)).
rule(context(c_demo), (true)).

%%%
%%% c_demo context
%%%

rule(modifier(moneyAmount, Object, currency, c_demo, Modifier),
      (cste(basic, Modifier, c_demo, "USD"))).

%%%
%%% dora context
%%%

rule(modifier(price, Object, includesServFee, dora, Modifier),
      (cste(basic, Modifier, dora, "yes"))).

rule(modifier(price, Object, includesVisaFee, dora, Modifier),
      (cste(basic, Modifier, dora, "no"))).

rule(modifier(price, Object, includesPaperCharge, dora, Modifier),
      (cste(basic, Modifier, dora, "yes"))).

rule(modifier(moneyAmount, Object, currency, dora, Modifier),
      (cste(basic, Modifier, dora, "USD"))).

%%%
%%% doras_friend context
%%%

rule(modifier(price, Object, includesServFee, doras_friend, Modifier),
      (cste(basic, Modifier, doras_friend, "no"))).

rule(modifier(price, Object, includesVisaFee, doras_friend, Modifier),
      (cste(basic, Modifier, doras_friend, "no"))).

rule(modifier(price, Object, includesPaperCharge, doras_friend, Modifier),
      (cste(basic, Modifier, doras_friend, "no"))).

rule(modifier(moneyAmount, Object, currency, doras_friend, Modifier),
      (cste(basic, Modifier, doras_friend, "GBP"))).

%%%
%%% c_yahoo context
%%%

rule(modifier(price, Object, includesServFee, c_yahoo, Modifier),
      (cste(basic, Modifier, c_yahoo, "yes"))).

rule(modifier(price, Object, includesVisaFee, c_yahoo, Modifier),
      (cste(basic, Modifier, c_yahoo, "no"))).

rule(modifier(price, Object, includesPaperCharge, c_yahoo, Modifier),
      (cste(basic, Modifier, c_yahoo, "no"))).

rule(modifier(moneyAmount, Object, currency, c_yahoo, Modifier),
      (cste(basic, Modifier, c_yahoo, "USD"))).

%%%

```

%% c_expedia context

%%

rule(modifier(price, Object, includesServFee, c_expedia, Modifier),
(cste(basic, Modifier, c_expedia, "yes"))).

rule(modifier(price, Object, includesVisaFee, c_expedia, Modifier),
(cste(basic, Modifier, c_expedia, "no"))).

rule(modifier(price, Object, includesPaperCharge, c_expedia, Modifier),
(cste(basic, Modifier, c_expedia, "no"))).

rule(modifier(moneyAmount, Object, currency, c_expedia, Modifier),
(cste(basic, Modifier, c_expedia, "USD"))).

%%

%% c_orbitz context

%%

rule(modifier(price, Object, includesServFee, c_orbitz, Modifier),
(cste(basic, Modifier, c_orbitz, "no"))).

rule(modifier(price, Object, includesVisaFee, c_orbitz, Modifier),
(cste(basic, Modifier, c_orbitz, "no"))).

rule(modifier(price, Object, includesPaperCharge, c_orbitz, Modifier),
(cste(basic, Modifier, c_orbitz, "no"))).

rule(modifier(moneyAmount, Object, currency, c_orbitz, Modifier),
(cste(basic, Modifier, c_orbitz, "USD"))).

%%

%% c_itn context

%%

rule(modifier(price, Object, includesServFee, c_itn, Modifier),
(cste(basic, Modifier, c_itn, "no"))).

rule(modifier(price, Object, includesVisaFee, c_itn, Modifier),
(cste(basic, Modifier, c_itn, "no"))).

rule(modifier(price, Object, includesPaperCharge, c_itn, Modifier),
(cste(basic, Modifier, c_itn, "no"))).

rule(modifier(moneyAmount, Object, currency, c_itn, Modifier),
(cste(basic, Modifier, c_itn, "USD"))).

%%

%% c_travelselect context

%%

rule(modifier(price, Object, includesServFee, c_travelselect, Modifier),
(cste(basic, Modifier, c_travelselect, "no"))).

rule(modifier(price, Object, includesVisaFee, c_travelselect, Modifier),
(cste(basic, Modifier, c_travelselect, "no"))).

rule(modifier(price, Object, includesPaperCharge, c_travelselect, Modifier),
(cste(basic, Modifier, c_travelselect, "no"))).

rule(modifier(moneyAmount, Object, currency, c_travelselect, Modifier),
(cste(basic, Modifier, c_travelselect, "GBP"))).

```

rule(attr(Price, provider, Provider), (myorbitz_p(_ Price, _ _ _ _ _ Provider, _))).
rule(attr(Price, provider, Provider), (yahoo_p(_ Price, _ _ _ _ _ Provider, _))).
rule(attr(Price, provider, Provider), (itn_p(_ Price, _ _ _ _ _ Provider))).
rule(attr(Price, provider, Provider), (expedia2_p(_ Price, _ _ _ _ _ Provider, _))).
rule(attr(Price, provider, Provider), (expedia_p(_ Price, _ _ _ _ _ Provider, _))).
rule(attr(Price, provider, Provider), (travelselect_p(_ Price, _ _ _ _ _ Provider, _))).

rule(attr(Price, destCntry, DestCntry), (myorbitz_p(_ Price, _ _ _ _ _ DestCntry))).
rule(attr(Price, destCntry, DestCntry), (yahoo_p(_ Price, _ _ _ _ _ DestCntry))).
rule(attr(Price, destCntry, DestCntry), (expedia2_p(_ Price, _ _ _ _ _ DestCntry))).
rule(attr(Price, destCntry, DestCntry), (expedia_p(_ Price, _ _ _ _ _ DestCntry))).
rule(attr(Price, destCntry, DestCntry), (travelselect_p(_ Price, _ _ _ _ _ DestCntry))).

rule(attr(Provider, serviceFee, Fee),
      (value(Provider, dora, P),
       value(ProviderSF, dora, P),
       'ServiceFees_p'(ProviderSF, Fee))).

rule(attr(Provider, paperTicketFee, Fee),
      (value(Provider, dora, P),
       value(ProviderSF, dora, P),
       paper_fees_p(ProviderSF, Fee))).

rule(attr(Destination, visaFees, VisaFee),
      (value(Destination, dora, D),
       value(DestinationF, dora, D),
       value(Citizenship, dora, "American"),
       visa_fees_p(Citizenship, DestinationF, VisaFee))).

rule(attr(X, txnDate, Y), (olsen_p(_6313, _6314, X, Y))).
rule(attr(X, fromCur, Y), (olsen_p(_6351, Y, X, _6354))).
rule(attr(X, toCur, Y), (olsen_p(Y, _6390, X, _6392))).

%%
%% conversion functions for price
%% with respect to priceType

rule(cvt(commutative, price, O, includesServFee, Ctxt, "no", Vs, "yes", Vt),
     (attr(O, provider, Pr),
      attr(Pr, serviceFee, Sf),
      value(Sf, Ctxt, Sfv),
      plus(Vs, Sfv, Vt))).

rule(cvt(commutative, price, O, includesVisaFee, Ctxt, "no", Vs, "yes", Vt),
     (attr(O, provider, Pr),
      attr(O, destCntry, De),
      attr(De, visaFees, Vf),
      value(Vf, Ctxt, VVf),
      plus(Vs, VVf, Vt))).

rule(cvt(commutative, price, O, includesPaperCharge, Ctxt, "no", Vs, "yes", Vt),
     (attr(O, provider, Pr),
      attr(Pr, paperTicketFee, Pf),
      value(Pf, Ctxt, Pfv),
      plus(Vs, Pfv, Vt))).

%% conversion functions for moneyAmount

```

%% with respect to currency
%%

```
rule(cvt(commuative, moneyAmount, _O, currency, Ctxt, Mvs, Vs, Mvt, Vt),
    (olsen_p(Fc, Tc, Rate, Date),
     value(Fc, Ctxt, Mvs),
     value(Tc, Ctxt, Mvt),
     value(Rate, Ctxt, Rv),
     currentDate_p(CurDate),
     value(CurDate, Ctxt, DateValue),
     value(Date, Ctxt, DateValue),
     multiply(Vs, Rv, Vt))).
```

```
rule(currentDate(Date), ({date(D), substring(D, 5, 3, Month), substring(D, 9, 2, Day), substring(D, 23, 2, Year)},
    month(Month, NumMonth), {concat_string([NumMonth, /, Day, /, Year], Date)})).
rule(month("Jan", 01), (true)).
rule(month("Feb", 02), (true)).
rule(month("Mar", 03), (true)).
rule(month("Apr", 04), (true)).
rule(month("May", 05), (true)).
rule(month("Jun", 06), (true)).
rule(month("Jul", 07), (true)).
rule(month("Aug", 08), (true)).
rule(month("Sep", 09), (true)).
rule(month("Oct", 10), (true)).
rule(month("Nov", 11), (true)).
rule(month("Dec", 12), (true)).
```

%%
%% elevations
%%

```
rule(currentDate_p(
    skolem(date, V, Ctxt, 1, currentDate(V))),
    (currentDate(V))).
```

```
rule(
    expedia_p(
        skolem(airline, C1, c_expedia, 1,
            expedia(C1, C2, C3, C4, C5, C6, C7, C8)),
        skolem(price, C2, c_expedia, 2,
            expedia(C1, C2, C3, C4, C5, C6, C7, C8)),
        skolem(cityORAIrport, C3, c_expedia, 3,
            expedia(C1, C2, C3, C4, C5, C6, C7, C8)),
        skolem(cityORAIrport, C4, c_expedia, 4,
            expedia(C1, C2, C3, C4, C5, C6, C7, C8)),
        skolem(date, C5, c_expedia, 5,
            expedia(C1, C2, C3, C4, C5, C6, C7, C8)),
        skolem(date, C6, c_expedia, 6,
            expedia(C1, C2, C3, C4, C5, C6, C7, C8)),
        skolem(provider, C7, c_expedia, 7,
            expedia(C1, C2, C3, C4, C5, C6, C7, C8)),
        skolem(country, C8, c_expedia, 8,
            expedia(C1, C2, C3, C4, C5, C6, C7, C8))),
    (expedia(C1, C2, C3, C4, C5, C6, C7, C8))).
```

```
rule(
    expedia2_p(
        %% skolem(trip, expediatrip, c_expedia, 9,
```

```

%%      expedia2(C1, C2, C3, C4, C5, C6, C7, C8)),
      skolem(airline, C1, c_expedia, 1,
        expedia2(C1, C2, C3, C4, C5, C6, C7, C8)),
      skolem(price, C2, c_expedia, 2,
        expedia2(C1, C2, C3, C4, C5, C6, C7, C8)),
      skolem(cityORAIrport, C3, c_expedia, 3,
        expedia2(C1, C2, C3, C4, C5, C6, C7, C8)),
      skolem(cityORAIrport, C4, c_expedia, 4,
        expedia2(C1, C2, C3, C4, C5, C6, C7, C8)),
      skolem(date, C5, c_expedia, 5,
        expedia2(C1, C2, C3, C4, C5, C6, C7, C8)),
      skolem(date, C6, c_expedia, 6,
        expedia2(C1, C2, C3, C4, C5, C6, C7, C8)),
      skolem(provider, C7, c_expedia, 7,
        expedia2(C1, C2, C3, C4, C5, C6, C7, C8)),
      skolem(country, C8, c_expedia, 8,
        expedia2(C1, C2, C3, C4, C5, C6, C7, C8))),
      (expedia2(C1, C2, C3, C4, C5, C6, C7, C8))).

```

```

rule(
  hotwire_p(
    skolem(null, C1, Ctxt, 1,
      hotwire(C1, C2, C3, C4, C5, C6, C7, C8)),
    skolem(null, C2, Ctxt, 2,
      hotwire(C1, C2, C3, C4, C5, C6, C7, C8)),
    skolem(null, C3, Ctxt, 3,
      hotwire(C1, C2, C3, C4, C5, C6, C7, C8)),
    skolem(null, C4, Ctxt, 4,
      hotwire(C1, C2, C3, C4, C5, C6, C7, C8)),
    skolem(null, C5, Ctxt, 5,
      hotwire(C1, C2, C3, C4, C5, C6, C7, C8)),
    skolem(null, C6, Ctxt, 6,
      hotwire(C1, C2, C3, C4, C5, C6, C7, C8)),
    skolem(null, C7, Ctxt, 7,
      hotwire(C1, C2, C3, C4, C5, C6, C7, C8)),
    skolem(null, C8, Ctxt, 8,
      hotwire(C1, C2, C3, C4, C5, C6, C7, C8))),
    (hotwire(C1, C2, C3, C4, C5, C6, C7, C8))).

```

```

rule(
  itn_p(
    skolem(airline, C1, c_itn, 1,
      itn(C1, C2, C3, C4, C5, C6, C7, C8, C9)),
    skolem(price, C2, c_itn, 2,
      itn(C1, C2, C3, C4, C5, C6, C7, C8, C9)),
    skolem(cityORAIrport, C3, c_itn, 3,
      itn(C1, C2, C3, C4, C5, C6, C7, C8, C9)),
    skolem(cityORAIrport, C4, c_itn, 4,
      itn(C1, C2, C3, C4, C5, C6, C7, C8, C9)),
    skolem(month, C5, c_itn, 5,
      itn(C1, C2, C3, C4, C5, C6, C7, C8, C9)),
    skolem(month, C6, c_itn, 6,
      itn(C1, C2, C3, C4, C5, C6, C7, C8, C9)),
    skolem(day, C7, c_itn, 7,
      itn(C1, C2, C3, C4, C5, C6, C7, C8, C9)),
    skolem(day, C8, c_itn, 8,
      itn(C1, C2, C3, C4, C5, C6, C7, C8, C9)),
    skolem(provider, C9, c_itn, 9,
      itn(C1, C2, C3, C4, C5, C6, C7, C8, C9))),
    (itn(C1, C2, C3, C4, C5, C6, C7, C8, C9))).

```

```

rule(
  myorbitz_p(
    skolem(airline, C1, c_orbitz, 1,
      myorbitz(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),
    skolem(price, C2, c_orbitz, 2,
      myorbitz(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),
    skolem(cityORAirport, C3, c_orbitz, 3,
      myorbitz(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),
    skolem(cityORAirport, C4, c_orbitz, 4,
      myorbitz(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),
    skolem(month, C5, c_orbitz, 5,
      myorbitz(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),
    skolem(month, C6, c_orbitz, 6,
      myorbitz(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),
    skolem(day, C7, c_orbitz, 7,
      myorbitz(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),
    skolem(day, C8, c_orbitz, 8,
      myorbitz(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),
    skolem(provider, C9, c_orbitz, 9,
      myorbitz(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),
    skolem(country, C10, c_orbitz, 10,
      myorbitz(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10))),
  (myorbitz(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10))).

```

```

rule(
  myunited_p(
    skolem(null, C1, Ctxt, 1,
      myunited(C1, C2, C3, C4, C5, C6, C7, C8)),
    skolem(null, C2, Ctxt, 2,
      myunited(C1, C2, C3, C4, C5, C6, C7, C8)),
    skolem(null, C3, Ctxt, 3,
      myunited(C1, C2, C3, C4, C5, C6, C7, C8)),
    skolem(null, C4, Ctxt, 4,
      myunited(C1, C2, C3, C4, C5, C6, C7, C8)),
    skolem(null, C5, Ctxt, 5,
      myunited(C1, C2, C3, C4, C5, C6, C7, C8)),
    skolem(null, C6, Ctxt, 6,
      myunited(C1, C2, C3, C4, C5, C6, C7, C8)),
    skolem(null, C7, Ctxt, 7,
      myunited(C1, C2, C3, C4, C5, C6, C7, C8)),
    skolem(null, C8, Ctxt, 8,
      myunited(C1, C2, C3, C4, C5, C6, C7, C8))),
  (myunited(C1, C2, C3, C4, C5, C6, C7, C8))).

```

```

rule(
  northwest_p(
    skolem(null, C1, Ctxt, 1,
      northwest(C1, C2, C3, C4, C5, C6, C7, C8)),
    skolem(null, C2, Ctxt, 2,
      northwest(C1, C2, C3, C4, C5, C6, C7, C8)),
    skolem(null, C3, Ctxt, 3,
      northwest(C1, C2, C3, C4, C5, C6, C7, C8)),
    skolem(null, C4, Ctxt, 4,
      northwest(C1, C2, C3, C4, C5, C6, C7, C8)),
    skolem(null, C5, Ctxt, 5,
      northwest(C1, C2, C3, C4, C5, C6, C7, C8)),
    skolem(null, C6, Ctxt, 6,
      northwest(C1, C2, C3, C4, C5, C6, C7, C8)),
    skolem(null, C7, Ctxt, 7,
      northwest(C1, C2, C3, C4, C5, C6, C7, C8)),
    skolem(null, C8, Ctxt, 8,
      northwest(C1, C2, C3, C4, C5, C6, C7, C8))),

```



```
northwest(C1, C2, C3, C4, C5, C6, C7, C8))),  
(northwest(C1, C2, C3, C4, C5, C6, C7, C8))).
```

```
rule(  
  paper_fees_p(  
    skolem(provider, C1, dora, 1,  
      paper_fees(C1, C2)),  
    skolem(paperTicketFee, C2, dora, 2,  
      paper_fees(C1, C2))),  
  (paper_fees(C1, C2))).
```

```
rule(  
  qixo_p(  
    skolem(null, C1, Ctxt, 1,  
      qixo(C1, C2, C3, C4, C5, C6)),  
    skolem(null, C2, Ctxt, 2,  
      qixo(C1, C2, C3, C4, C5, C6)),  
    skolem(null, C3, Ctxt, 3,  
      qixo(C1, C2, C3, C4, C5, C6)),  
    skolem(null, C4, Ctxt, 4,  
      qixo(C1, C2, C3, C4, C5, C6)),  
    skolem(null, C5, Ctxt, 5,  
      qixo(C1, C2, C3, C4, C5, C6)),  
    skolem(null, C6, Ctxt, 6,  
      qixo(C1, C2, C3, C4, C5, C6))),  
  (qixo(C1, C2, C3, C4, C5, C6))).
```

```
rule(  
  'ServiceFees_p'(  
    skolem(provider, C1, dora, 1,  
      'ServiceFees'(C1, C2)),  
    skolem(serviceFee, C2, dora, 2,  
      'ServiceFees'(C1, C2))),  
  ('ServiceFees'(C1, C2))).
```

```
rule(  
  travelocity_p(  
    skolem(null, C1, Ctxt, 1,  
      travelocity(C1, C2, C3, C4, C5, C6, C7, C8)),  
    skolem(null, C2, Ctxt, 2,  
      travelocity(C1, C2, C3, C4, C5, C6, C7, C8)),  
    skolem(null, C3, Ctxt, 3,  
      travelocity(C1, C2, C3, C4, C5, C6, C7, C8)),  
    skolem(null, C4, Ctxt, 4,  
      travelocity(C1, C2, C3, C4, C5, C6, C7, C8)),  
    skolem(null, C5, Ctxt, 5,  
      travelocity(C1, C2, C3, C4, C5, C6, C7, C8)),  
    skolem(null, C6, Ctxt, 6,  
      travelocity(C1, C2, C3, C4, C5, C6, C7, C8)),  
    skolem(null, C7, Ctxt, 7,  
      travelocity(C1, C2, C3, C4, C5, C6, C7, C8)),  
    skolem(null, C8, Ctxt, 8,  
      travelocity(C1, C2, C3, C4, C5, C6, C7, C8))),  
  (travelocity(C1, C2, C3, C4, C5, C6, C7, C8))).
```

```
rule(  
  visa_fees_p(  
    skolem(country, C1, dora, 1,  
      visa_fees(C1, C2, C3)),  
    skolem(country, C2, dora, 2,  
      visa_fees(C1, C2, C3))),
```

```
skolem(visaFees, C3, dora, 3,  
    visa_fees(C1, C2, C3))),  
(visa_fees(C1, C2, C3))).
```

```
rule(  
    olsen_p(  
        skolem(currency, Exch, olsen_context, 1,  
            olsen(Exch, Express, Rate, Date)),  
        skolem(currency, Express, olsen_context, 2,  
            olsen(Exch, Express, Rate, Date)),  
        skolem(exchangeRate, Rate, olsen_context, 3,  
            olsen(Exch, Express, Rate, Date)),  
        skolem(date, Date, olsen_context, 4,  
            olsen(Exch, Express, Rate, Date))),  
    (olsen(Exch, Express, Rate, Date))).
```

```
rule(  
    yahoo_p(  
        skolem(airline, C1, c_yahoo, 1,  
            yahoo(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),  
        skolem(price, C2, c_yahoo, 2,  
            yahoo(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),  
        skolem(cityORAIrport, C3, c_yahoo, 3,  
            yahoo(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),  
        skolem(cityORAIrport, C4, c_yahoo, 4,  
            yahoo(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),  
        skolem(month, C5, c_yahoo, 5,  
            yahoo(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),  
        skolem(month, C6, c_yahoo, 6,  
            yahoo(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),  
        skolem(day, C7, c_yahoo, 7,  
            yahoo(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),  
        skolem(day, C8, c_yahoo, 8,  
            yahoo(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),  
        skolem(provider, C9, c_yahoo, 9,  
            yahoo(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),  
        skolem(country, C10, c_yahoo, 10,  
            yahoo(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10))),  
    (yahoo(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10))).
```

```
rule(  
    travelselect_p(  
        skolem(airline, C1, c_travelselect, 1,  
            travelselect(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),  
        skolem(price, C2, c_travelselect, 2,  
            travelselect(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),  
        skolem(cityORAIrport, C3, c_travelselect, 3,  
            travelselect(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),  
        skolem(cityORAIrport, C4, c_travelselect, 4,  
            travelselect(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),  
        skolem(month, C5, c_travelselect, 5,  
            travelselect(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),  
        skolem(month, C6, c_travelselect, 6,  
            travelselect(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),  
        skolem(day, C7, c_travelselect, 7,  
            travelselect(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),  
        skolem(day, C8, c_travelselect, 8,  
            travelselect(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),  
        skolem(provider, C9, c_travelselect, 9,  
            travelselect(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10)),  
        skolem(country, C10, c_travelselect, 10,
```

```
travelselect(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10))),
(travelselect(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10))).
```

```
%%  
%% Relations  
%%
```

```
rule(relation(cameleon,  
expedia,  
ie,  
[['Airline', string],  
['Price', string],  
['Destination', string],  
['Departure', string],  
['Date1', string],  
['Date2', string],  
['Provider', string],  
['IsIn', string]],  
cap([[0, 0, 0, 0, 0, 0, 0, 0]],  
[]), (true)).
```

```
rule(relation(cameleon,  
expedia2,  
ie,  
[['Airline', string],  
['Price', string],  
['Destination', string],  
['Departure', string],  
['Date1', string],  
['Date2', string],  
['Provider', string],  
['IsIn', string]],  
cap([[0, 0, 0, 0, 0, 0, 0, 0]],  
[]), (true)).
```

```
rule(relation(cameleon,  
hotwire,  
ie,  
[['Airline', string],  
['Price', string],  
['Destination', string],  
['Departure', string],  
[month1, string],  
[month2, string],  
[day1, string],  
[day2, string]],  
cap([[0, 0, 0, 0, 0, 0, 0, 0]],  
[]), (true)).
```

```
rule(relation(cameleon,  
itn,  
ie,  
[['Airline', string],  
['Price', string],  
['Destination', string],  
['Departure', string],  
['Month1', string],  
['Month2', string],  
['Day1', string],  
['Day2', string],  
['Provider', string]],
```

```

cap([[0, 0, 0, 0, 0, 0, 0, 0, 0]],
[]), (true)).

rule(relation(cameleon,
myorbitz,
ie,
[['Airline', string],
['Price', string],
['Destination', string],
['Departure', string],
['Month1', string],
['Month2', string],
['Day1', string],
['Day2', string],
['Provider', string],
['IsIn', string]],
cap([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],
[]), (true)).

rule(relation(cameleon,
myunited,
ie,
[['Airline', string],
['Price', string],
['Destination', string],
['Departure', string],
['Month1', string],
['Month2', string],
['Day1', string],
['Day2', string]],
cap([[0, 0, 0, 0, 0, 0, 0, 0, 0]],
[]), (true)).

rule(relation(cameleon,
northwest,
ie,
[['Airline', string],
['Price', string],
['Destination', string],
['Departure', string],
['Month1', string],
['Month2', string],
['Day1', string],
['Day2', string]],
cap([[0, 0, 0, 0, 0, 0, 0, 0, 0]],
[]), (true)).

rule(relation(oracle,
paper_fees,
i,
[[provider, string],
[paperfee, string]],
cap([[0, 0]],
[]), (true)).

rule(relation(cameleon,
qixo,
ie,
[['Airline', string],
['Price', string],
['Destination', string],

```

```

    ['Departure', string],
    [date1, string],
    [date2, string]],
    cap([[0, 0, 0, 0, 0, 0]],
    []), (true)).

rule(relation(oracle,
    'ServiceFees',
    i,
    [['Provider', string],
    ['ServiceFee', number]],
    cap([[0, 0]],
    []), (true)).

rule(relation(cameleon,
    olsen,
    ie,
    [['Exchanged', string],
    ['Expressed', string],
    ['Rate', real],
    ['Date', string]],
    cap([[0, 0, 0, 0]],
    []), (true)).

rule(relation(cameleon,
    travelocity,
    ie,
    [['Airline', string],
    ['Price', string],
    ['Destination', string],
    ['Departure', string],
    ['Month1', string],
    ['Month2', string],
    ['Day1', string],
    ['Day2', string]],
    cap([[0, 0, 0, 0, 0, 0, 0, 0]],
    []), (true)).

rule(relation(oracle,
    visa_fees,
    i,
    [['citizenship', string],
    ['destination', string],
    ['visafee', string]],
    cap([[0, 0, 0]],
    []), (true)).

rule(relation(cameleon,
    yahoo,
    ie,
    [['Airline', string],
    ['Price', string],
    ['Destination', string],
    ['Departure', string],
    ['Month1', string],
    ['Month2', string],
    ['Day1', string],
    ['Day2', string],
    ['Provider', string],
    ['IsIn', string]],
    cap([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],

```

```
    []), (true)).  
rule(relation(cameleon,  
  travelselect,  
  ie,  
  [['Airline', string],  
  ['Price', string],  
  ['Destination', string],  
  ['Departure', string],  
  ['Month1', string],  
  ['Month2', string],  
  ['Day1', string],  
  ['Day2', string],  
  ['Provider', string],  
  ['IsIn', string]],  
  cap([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],  
    []), (true)).
```

Appendix B – Application Prolog file from Car Rental

```
:- module_interface(application513).
:- export rule/2.
:- begin_module(application513).
:- dynamic rule/2.

%%% domain model for Car Renter
%%% generation timestamp: 4/21/2003 10:20:30 PM

%%%
%%% Semantic types
%%%
rule(is_a(airportCode, basic), (true)).
rule(is_a(city, basic), (true)).
rule(is_a(company, basic), (true)).
rule(is_a(date2, basic), (true)).
rule(is_a(day2, basic), (true)).
rule(is_a(divideFactor, basic), (true)).
rule(is_a(moneyAmount2, basic), (true)).
rule(is_a(month2, basic), (true)).
rule(is_a(price2, moneyAmount2), (true)).
rule(is_a(ratePeriod, basic), (true)).
rule(is_a(rental, basic), (true)).
rule(is_a(joe, basic), (true)).
rule(is_a(joes_friend, basic), (true)).
rule(is_a(c_expediacar, basic), (true)).
rule(is_a(c_yahoocar, basic), (true)).
rule(is_a(c_qixocar, basic), (true)).

rule(contexts([joe, joes_friend, c_expediacar, c_yahoocar, c_qixocar]),(true)).

%%%
%%% Modifiers
%%%
rule(modifiers(airportCode, []), (true)).
rule(modifiers(basic, []), (true)).
rule(modifiers(city, [airportOrLocation]), (true)).
rule(modifiers(company, []), (true)).
rule(modifiers(date2, []), (true)).
rule(modifiers(day2, []), (true)).
rule(modifiers(divideFactor, []), (true)).
rule(modifiers(moneyAmount2, []), (true)).
rule(modifiers(month2, [month2SymbolType]), (true)).
%%%rule(modifiers(price2, [price2Type]), (true)).
rule(modifiers(price2, [includesServiceFee, includesServiceFeeE, includesServiceFeeQ, includesServiceFeeY]), (true)).
rule(modifiers(ratePeriod, []), (true)).
rule(modifiers(rental, []), (true)).

%%%
%%% Attributes
%%%
rule(attributes(airportCode, []), (true)).
rule(attributes(basic, []), (true)).
rule(attributes(city, [airportCode]), (true)).
rule(attributes(company, []), (true)).
rule(attributes(date2, [hasday2, hasmonth2]), (true)).
rule(attributes(day2, []), (true)).
rule(attributes(divideFactor, []), (true)).
rule(attributes(moneyAmount2, []), (true)).
```

```

rule(attributes(month2, []), (true)).
rule(attributes(price2, [dropoffday2, dropoffmonth2, pickupday2, pickupmonth2, ratePeriod]), (true)).
rule(attributes(ratePeriod, [divideFactor]), (true)).
rule(attributes(rental, [dropoff, dropoffday2, dropoffmonth2, pickup, pickupday2, pickupmonth2, price2, ratePeriod,
rentalCompany]), (true)).

%%%
%%% Contexts
%%%
rule(context(joe), (true)).
rule(context(joes_friend), (true)).
rule(context(c_expediacar), (true)).
rule(context(c_yahoocar), (true)).
rule(context(c_qixocar), (true)).

%%%
%%% joe context
%%%
rule(modifier(price2, Object, price2Type, c_PriceByRate, Modifier),
      (cste(basic, Modifier, c_PriceByRate, "price2AsRate"))).

rule(modifier(price2, Object, includesServiceFee, joe, Modifier),
      (cste(basic, Modifier, joe, "yes"))).

%%%rule(modifier(price2, Object, includesServiceFeeQ, joe, Modifier),
%%%      (cste(basic, Modifier, joe, "yes"))).

%%%rule(modifier(price2, Object, includesServiceFeeY, joe, Modifier),
%%%      (cste(basic, Modifier, joe, "yes"))).

rule(modifier(month2, Object, month2SymbolType, joe, Modifier),
      (cste(basic, Modifier, joe, "threeLetter"))).

rule(modifier(city, Object, airportOrLocation, joe, Modifier),
      (cste(basic, Modifier, joe, "location"))).

%%%
%%% joes_friend context
%%%
rule(modifier(price2, Object, price2Type, c_PriceByRate, Modifier),
      (cste(basic, Modifier, c_PriceByRate, "price2AsRate"))).

rule(modifier(price2, Object, includesServiceFee, joes_friend, Modifier),
      (cste(basic, Modifier, joes_friend, "yes"))).

%%%rule(modifier(price2, Object, includesServiceFeeQ, joes_friend, Modifier),
%%%      (cste(basic, Modifier, joes_friend, "yes"))).

%%%rule(modifier(price2, Object, includesServiceFeeY, joes_friend, Modifier),
%%%      (cste(basic, Modifier, joes_friend, "yes"))).

rule(modifier(month2, Object, month2SymbolType, joes_friend, Modifier),
      (cste(basic, Modifier, joes_friend, "numeric"))).

rule(modifier(city, Object, airportOrLocation, joes_friend, Modifier),
      (cste(basic, Modifier, joes_friend, "airport"))).

%%%
%%% c_yahoocar context
%%%
rule(modifier(price2, Object, includesServiceFee, c_yahoocar, Modifier),

```



```

(cste(basic, Modifier, c_yahoocar, "yes"))).

%%rule(modifier(price2, Object, includesServiceFeeQ, c_yahoocar, Modifier),
%% (cste(basic, Modifier, c_yahoocar, "no"))).

%%rule(modifier(price2, Object, includesServiceFeeY, c_yahoocar, Modifier),
%% (cste(basic, Modifier, c_yahoocar, "yes"))).

rule(modifier(month2, Object, month2SymbolType, c_yahoocar, Modifier),
(cste(basic, Modifier, c_yahoocar, "threeLetter"))).

rule(modifier(city, Object, airportOrLocation, c_yahoocar, Modifier),
(cste(basic, Modifier, c_yahoocar, "airport"))).

%%
%% c_expediacar context
%%
rule(modifier(price2, Object, includesServiceFee, c_expediacar, Modifier),
(cste(basic, Modifier, c_expediacar, "yes"))).

%%rule(modifier(price2, Object, includesServiceFeeQ, c_expediacar, Modifier),
%% (cste(basic, Modifier, c_expediacar, "no"))).

%%rule(modifier(price2, Object, includesServiceFeeY, c_expediacar, Modifier),
%% (cste(basic, Modifier, c_expediacar, "no"))).

rule(modifier(month2, Object, month2SymbolType, c_expediacar, Modifier),
(cste(basic, Modifier, c_expediacar, "numeric"))).

rule(modifier(city, Object, airportOrLocation, c_expediacar, Modifier),
(cste(basic, Modifier, c_expediacar, "airport"))).

%%
%% c_qixocar context
%%
rule(modifier(price2, Object, includesServiceFee, c_qixocar, Modifier),
(cste(basic, Modifier, c_qixocar, "no"))).

%%rule(modifier(price2, Object, includesServiceFeeQ, c_qixocar, Modifier),
%% (cste(basic, Modifier, c_qixocar, "no"))).

%%rule(modifier(price2, Object, includesServiceFeeY, c_qixocar, Modifier),
%% (cste(basic, Modifier, c_qixocar, "no"))).

rule(modifier(month2, Object, month2SymbolType, c_qixocar, Modifier),
(cste(basic, Modifier, c_qixocar, "numeric"))).

rule(modifier(city, Object, airportOrLocation, c_qixocar, Modifier),
(cste(basic, Modifier, c_qixocar, "airport"))).

rule(attr(Price2, ratePeriod, RatePeriod), (expediacar_p(, , , , , Price2, , RatePeriod))).
rule(attr(Price2, ratePeriod, RatePeriod), (yahoocar_p(, , , , , Price2, , RatePeriod))).
rule(attr(Price2, ratePeriod, RatePeriod), (orbitzcar_p(, , , , , Price2, , RatePeriod))).
rule(attr(Price2, ratePeriod, RatePeriod), (qixocar_p(, , , , , Price2, , RatePeriod))).

rule(attr(Price2, pickupmonth2, Pickupmonth2), (expediacar_p(, , , , , Pickupmonth2, , , Price2, , _))).
rule(attr(Price2, pickupmonth2, Pickupmonth2), (yahoocar_p(, , , , , Pickupmonth2, , , Price2, , _))).
rule(attr(Price2, pickupmonth2, Pickupmonth2), (orbitzcar_p(, , , , , Pickupmonth2, , , Price2, , _))).

```

```

rule(attr(Price2, pickupmonth2, Pickupmonth2), (qixocar_p(, , Pickupmonth2, , , Price2, , _))).

rule(attr(Price2, dropoffmonth2, Dropoffmonth2), (expediacar_p(, , , , Dropoffmonth2, , , Price2, , _))).
rule(attr(Price2, dropoffmonth2, Dropoffmonth2), (yahoocar_p(, , , Dropoffmonth2, , , Price2, , _))).
rule(attr(Price2, dropoffmonth2, Dropoffmonth2), (orbitzcar_p(, , , Dropoffmonth2, , , Price2, , _))).
rule(attr(Price2, dropoffmonth2, Dropoffmonth2), (qixocar_p(, , , Dropoffmonth2, , , Price2, , _))).

rule(attr(Price2, pickupday2, Pickupday2), (expediacar_p(, , , , , Pickupday2, , , Price2, , _))).
rule(attr(Price2, pickupday2, Pickupday2), (yahoocar_p(, , , , , Pickupday2, , , Price2, , _))).
rule(attr(Price2, pickupday2, Pickupday2), (orbitzcar_p(, , , , , Pickupday2, , , Price2, , _))).
rule(attr(Price2, pickupday2, Pickupday2), (qixocar_p(, , , , , Pickupday2, , , Price2, , _))).

rule(attr(Price2, dropoffday2, Dropoffday2), (expediacar_p(, , , , , , Dropoffday2, Price2, , _))).
rule(attr(Price2, dropoffday2, Dropoffday2), (yahoocar_p(, , , , , , Dropoffday2, Price2, , _))).
rule(attr(Price2, dropoffday2, Dropoffday2), (orbitzcar_p(, , , , , , Dropoffday2, Price2, , _))).
rule(attr(Price2, dropoffday2, Dropoffday2), (qixocar_p(, , , , , , Dropoffday2, Price2, , _))).

rule(attr(RatePeriod, divideFactor, DivideFactor),
      (value(RatePeriod, c_PriceByRate, R),
       value(RatePeriodF, c_PriceByRate, R),
       rate_period_dividefactors_p(RatePeriodF, DivideFactor))).

rule(attr(Cityname, airportCode, Airportcode),
      (value(Cityname, c_PriceByRate, C),
       value(CitynameF, c_PriceByRate, C),
       airport_code_lookup_p(CitynameF, Airportcode))).

%%
%% conversion functions for month2
%% with respect to month2SymbolType

rule(cvt(, month2, O, month2SymbolType, Ctxt, "numeric", Vs, "threeLetter", Vt),
      (month_symbol_converter(Vs, Vt))).

rule(cvt(, month2, O, month2SymbolType, Ctxt, "threeLetter", Vs, "numeric", Vt),
      (month_symbol_converter(Vt, Vs))).

%%
%% conversion functions for city
%% with respect to airportOrLocation

rule(cvt(, city, O, airportOrLocation, Ctxt, "location", Vs, "airport", Vt),
      (airport_code_lookup(Vs, Vt))).

rule(cvt(, city, O, airportOrLocation, Ctxt, "airport", Vs, "location", Vt),
      (airport_code_lookup(Vt, Vs))).

%%
%% conversion functions for price2
%% with respect to includesServiceFee

rule(cvt(commutative, price2, O, includesServiceFee, Ctxt, "no", Vs, "yes", Vt),
      (plus(Vs, 9.99, Vt))).

rule(cvt(commutative, price2, O, includesServiceFeeY, Ctxt, "no", Vs, "yes", Vt),
      (plus(Vs, 5, Vt))).

rule(cvt(commutative, price2, O, includesServiceFeeQ, Ctxt, "no", Vs, "yes", Vt),

```

```

(plus(Vs, 10, Vt)).

%%
%% elevations
%%

rule(
  airport_code_lookup_p(
    skolem(city, C1, Ctxt, 1,
      airport_code_lookup(C1, C2)),
    skolem(city, C2, Ctxt, 2,
      airport_code_lookup(C1, C2))),
  (airport_code_lookup(C1, C2))).

rule(
  expediacar_p(
    skolem(city, C1, c_expediacar, 1,
      expediacar(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11)),
    skolem(city, C2, c_expediacar, 2,
      expediacar(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11)),
    skolem(date2, C3, c_expediacar, 3,
      expediacar(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11)),
    skolem(date2, C4, c_expediacar, 4,
      expediacar(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11)),
    skolem(month2, C5, c_expediacar, 5,
      expediacar(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11)),
    skolem(month2, C6, c_expediacar, 6,
      expediacar(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11)),
    skolem(day2, C7, c_expediacar, 7,
      expediacar(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11)),
    skolem(day2, C8, c_expediacar, 8,
      expediacar(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11)),
    skolem(price2, C9, c_expediacar, 9,
      expediacar(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11)),
    skolem(company, C10, c_expediacar, 10,
      expediacar(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11)),
    skolem(ratePeriod, C11, c_expediacar, 11,
      expediacar(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11))),
  (expediacar(C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11))).

%%rule(
%% orbitzcar_p(
%% ),
%% (orbitzcar())).

rule(
  month_symbol_converter_p(
    skolem(month2, C1, Ctxt, 1,
      month_symbol_converter(C1, C2)),
    skolem(month2, C2, Ctxt, 2,
      month_symbol_converter(C1, C2))),
  (month_symbol_converter(C1, C2))).

rule(
  rate_period_dividefactors_p(
    skolem(ratePeriod, C1, Ctxt, 1,
      rate_period_dividefactors(C1, C2)),
    skolem(divideFactor, C2, Ctxt, 2,
      rate_period_dividefactors(C1, C2))),
  (rate_period_dividefactors(C1, C2))).

```

```

rule(
  yahooocar_p(
    skolem(city, C1, c_yahooocar, 1,
      yahooocar(C1, C2, C3, C4, C5, C6, C7, C8, C9)),
    skolem(city, C2, c_yahooocar, 2,
      yahooocar(C1, C2, C3, C4, C5, C6, C7, C8, C9)),
    skolem(month2, C3, c_yahooocar, 3,
      yahooocar(C1, C2, C3, C4, C5, C6, C7, C8, C9)),
    skolem(month2, C4, c_yahooocar, 4,
      yahooocar(C1, C2, C3, C4, C5, C6, C7, C8, C9)),
    skolem(day2, C5, c_yahooocar, 5,
      yahooocar(C1, C2, C3, C4, C5, C6, C7, C8, C9)),
    skolem(day2, C6, c_yahooocar, 6,
      yahooocar(C1, C2, C3, C4, C5, C6, C7, C8, C9)),
    skolem(price2, C7, c_yahooocar, 7,
      yahooocar(C1, C2, C3, C4, C5, C6, C7, C8, C9)),
    skolem(company, C8, c_yahooocar, 8,
      yahooocar(C1, C2, C3, C4, C5, C6, C7, C8, C9)),
    skolem(ratePeriod, C9, c_yahooocar, 9,
      yahooocar(C1, C2, C3, C4, C5, C6, C7, C8, C9))),
  (yahooocar(C1, C2, C3, C4, C5, C6, C7, C8, C9))).

```

```

rule(
  qixocar_p(
    skolem(city, C1, c_qixocar, 1,
      qixocar(C1, C2, C3, C4, C5, C6, C7, C8, C9)),
    skolem(city, C2, c_qixocar, 2,
      qixocar(C1, C2, C3, C4, C5, C6, C7, C8, C9)),
    skolem(month2, C3, c_qixocar, 3,
      qixocar(C1, C2, C3, C4, C5, C6, C7, C8, C9)),
    skolem(month2, C4, c_qixocar, 4,
      qixocar(C1, C2, C3, C4, C5, C6, C7, C8, C9)),
    skolem(day2, C5, c_qixocar, 5,
      qixocar(C1, C2, C3, C4, C5, C6, C7, C8, C9)),
    skolem(day2, C6, c_qixocar, 6,
      qixocar(C1, C2, C3, C4, C5, C6, C7, C8, C9)),
    skolem(price2, C7, c_qixocar, 7,
      qixocar(C1, C2, C3, C4, C5, C6, C7, C8, C9)),
    skolem(company, C8, c_qixocar, 8,
      qixocar(C1, C2, C3, C4, C5, C6, C7, C8, C9)),
    skolem(ratePeriod, C9, c_qixocar, 9,
      qixocar(C1, C2, C3, C4, C5, C6, C7, C8, C9))),
  (qixocar(C1, C2, C3, C4, C5, C6, C7, C8, C9))).

```

```

%%
%% Relations
%%

```

```

rule(relation(oracle,
  airport_code_lookup,
  ie,
  [[location, string],
  [airportcode, string]],
  cap([[0, 0]],
  [])), (true)).

```

```

rule(relation(cameleon,
  expediacar,
  ie,

```

```

[[['Pickup', string],
 ['Dropoff', string],
 ['Date1', string],
 ['Date2', string],
 ['Month1', string],
 ['Month2', string],
 ['Day1', string],
 ['Day2', string],
 ['Price', string],
 ['Company', string],
 ['Rateperiod', string]],
 cap([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],
 [])], (true)).

```

```

rule(relation(cameleon,
 orbitzcar,
 ie,
 []),
 cap([[[]],
 []]), (true)).

```

```

rule(relation(oracle,
 month_symbol_converter,
 ie,
 [[mm_number, string],
 [symbol, string]],
 cap([[0, 0]],
 []]), (true)).

```

```

rule(relation(oracle,
 rate_period_dividefactors,
 ie,
 [['Rateperiod', string],
 ['Dividefactor', string]],
 cap([[0, 0]],
 []]), (true)).

```

```

rule(relation(cameleon,
 yahoocar,
 ie,
 [['Pickup', string],
 ['Dropoff', string],
 ['Month1', string],
 ['Month2', string],
 ['Day1', string],
 ['Day2', string],
 ['Price', string],
 ['Company', string],
 ['Rateperiod', string]],
 cap([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],
 []]), (true)).

```

```

rule(relation(cameleon,
 qixocar,
 ie,
 [['Pickup', string],
 ['Dropoff', string],
 ['Month1', string],
 ['Month2', string],
 ['Day1', string],
 ['Day2', string],

```

```
['Price', string],
['Company', string],
['Rateperiod', string]],
cap([[0, 0, 0, 0, 0, 0, 0, 0, 0]],
[]), (true)).
```

Appendix C – Merger Axioms file for Airfare + Car Rental Merger

```
:- module_interface(application514).
:- export rule/2.
:- begin_module(application514).
:- dynamic rule/2.

%%% 512: Airfare, 513: Car Renter, 514: Merged Travel

rule(merges([application512,application513]),(true)).

rule(contexts([c_FlyAndRent]),(true)).

rule(isomodifiertypes(application514,application513,cityORAirport,city), (true)).
rule(isomodifiertypes(application514,application513,month,month2), (true)).
rule(isomodifiertypes(application514,application513,day,day2), (true)).
rule(isomodifiertypes(application514,application512,moneyAmount2,moneyAmount), (true)).

rule(modifiers(cityORAirport, [airportOrLocation]), (true)).
rule(modifiers(month, [month2SymbolType]), (true)).
rule(modifiers(moneyAmount2, [currency]), (true)).
rule(modifiers(price, [includesCarRental]), (true)).
rule(modifiers(provider, [includesCarCompany]), (true)).

rule(modifier(cityORAirport, O, airportOrLocation, dora, M), (cste(basic, M, dora, "location"))).
rule(modifier(cityORAirport, O, airportOrLocation, doras_friend, M), (cste(basic, M, doras_friend, "location"))).
rule(modifier(cityORAirport, O, airportOrLocation, c_yahoo, M), (cste(basic, M, c_yahoo, "airport"))).
rule(modifier(cityORAirport, O, airportOrLocation, c_expedia, M), (cste(basic, M, c_expedia, "airport"))).
rule(modifier(cityORAirport, O, airportOrLocation, c_orbitz, M), (cste(basic, M, c_orbitz, "airport"))).
rule(modifier(cityORAirport, O, airportOrLocation, c_itn, M), (cste(basic, M, c_itn, "airport"))).
rule(modifier(cityORAirport, O, airportOrLocation, c_travelselect, M), (cste(basic, M, c_travelselect, "airport"))).

rule(modifier(month, O, month2SymbolType, dora, M), (cste(basic, M, dora, "threeLetter"))).
rule(modifier(month, O, month2SymbolType, doras_friend, M), (cste(basic, M, doras_friend, "numeric"))).
rule(modifier(month, O, month2SymbolType, c_yahoo, M), (cste(basic, M, c_yahoo, "threeLetter"))).
rule(modifier(month, O, month2SymbolType, c_expedia, M), (cste(basic, M, c_expedia, "numeric"))).
rule(modifier(month, O, month2SymbolType, c_orbitz, M), (cste(basic, M, c_orbitz, "threeLetter"))).
rule(modifier(month, O, month2SymbolType, c_itn, M), (cste(basic, M, c_itn, "threeLetter"))).
rule(modifier(month, O, month2SymbolType, c_travelselect, M), (cste(basic, M, c_travelselect, "numeric"))).

rule(modifier(moneyAmount2, O, currency, joe, M), (cste(basic, M, joe, "USD"))).
rule(modifier(moneyAmount2, O, currency, c_expediacar, M), (cste(basic, M, c_expediacar, "USD"))).
rule(modifier(moneyAmount2, O, currency, c_yahoocar, M), (cste(basic, M, c_yahoocar, "USD"))).
rule(modifier(moneyAmount2, O, currency, c_qixocar, M), (cste(basic, M, c_qixocar, "USD"))).

rule(modifier(price, Object, includesCarRental, dora, Modifier), (cste(basic, Modifier, dora, "no"))).
rule(modifier(price, Object, includesCarRental, doras_friend, Modifier), (cste(basic, Modifier, doras_friend, "no"))).
rule(modifier(price, Object, includesCarRental, c_expedia, Modifier), (cste(basic, Modifier, c_expedia, "no"))).
rule(modifier(price, Object, includesCarRental, c_yahoo, Modifier), (cste(basic, Modifier, c_yahoo, "no"))).
rule(modifier(price, Object, includesCarRental, c_itn, Modifier), (cste(basic, Modifier, c_itn, "no"))).
rule(modifier(price, Object, includesCarRental, c_orbitz, Modifier), (cste(basic, Modifier, c_orbitz, "no"))).
rule(modifier(price, Object, includesCarRental, c_travelselect, Modifier), (cste(basic, Modifier, c_travelselect, "no"))).

rule(modifier(provider, O, includesCarCompany, dora, M), (cste(basic, M, dora, "no"))).
rule(modifier(provider, O, includesCarCompany, doras_friend, M), (cste(basic, M, doras_friend, "no"))).
rule(modifier(provider, O, includesCarCompany, c_yahoo, M), (cste(basic, M, c_yahoo, "no"))).
rule(modifier(provider, O, includesCarCompany, c_expedia, M), (cste(basic, M, c_expedia, "no"))).
rule(modifier(provider, O, includesCarCompany, c_orbitz, M), (cste(basic, M, c_orbitz, "no"))).
rule(modifier(provider, O, includesCarCompany, c_itn, M), (cste(basic, M, c_itn, "no"))).
rule(modifier(provider, O, includesCarCompany, c_travelselect, M), (cste(basic, M, c_travelselect, "no"))).
```


plus(Vs, Rpv, Vt)).

```
rule(cvt(commutative, provider, Obj, includesCarCompany, Ctxt, "no", Vs, "yes", Vt),
(yahoocar_p(DestC, Dropoff, M1C, M2C, D1C, D2C, _, Comp, _),
attr(O, provider, Obj),
attr(O, month1, M1),
attr(O, month2, M2),
attr(O, day1, D1),
attr(O, day2, D2),
attr(O, destination, Dest),
value(M1, c_yahoo, M1v),
value(M1C, c_yahoo, M1v),
value(M2, c_yahoo, M2v),
value(M2C, c_yahoo, M2v),
value(D1, c_yahoo, D1v),
value(D1C, c_yahoo, D1v),
value(D2, c_yahoo, D2v),
value(D2C, c_yahoo, D2v),
value(Dest, c_yahoo, Destv),
value(DestC, c_yahoo, Destv),
value(Dropoff, c_yahoo, "same"),
value(Comp, Ctxt, Compv),
concat(Vs, "&", Vt1),
concat(Vt1, Compv, Vt))).
```

Appendix D – Additions to API to Support CLAMP’s Changes to ICM

This appendix contains the additions need in the ICM API to support merging. To see the full API for the ICM in Application Editor, see Appendix A in [12].

```
public void createIsoModifierType (string newIsoModifierTypeName,  
                                   Ont_semanticType semanticType1, string App1,  
                                   Ont_semanticType semanticType2, string App2)  
  
//Parameters: newIsoModifierTypeName - the name given to the isomodifiertype in the  
merged application; semanticType1, semanticType2 - the semanticTypes that are being  
pulled up in the new merged application; App1, App2 - the applications that the  
linked semantic types are coming from.  
  
//Remarks: creates a new isomodifiertype, pulling up associated modifiers from each  
semantic type.  
  
public void createIsoModifier (string newIsoModifierName,  
                               Ont_Modifier modifier1, string App1,  
                               Ont_Modifier modifier2, string App2)  
  
//Parameters: newIsoModifierName - the name given to the isoModifier in the merged  
application; modifier1, modifier2 - the modifiers that are being pulled up in the  
new merged application; App1, App2 - the applications that the linked modifiers are  
coming from.  
  
//Remarks: creates a new isoModifier.  
  
public void createIsoAttribute (string newIsoAttributeName,  
                                Ont_Attribute attribute1, string App1,  
                                Ont_Attribute attribute2, string App2)  
  
//Parameters: newIsoAttributeName - the name given to the isoAttribute in the  
merged application; attribute1, attribute2 - the attributes that are being pulled  
up in the new merged application; App1, App2 - the applications that the linked  
attributes are coming from.  
  
//Remarks: creates a new isoAttribute.  
  
public void createIsoContext(string newIsoContextName, Cxt_context context1,  
                             string App1, Cxt_context context2, string App2)  
  
//Parameters: newIsoContextName - the name given to the isoContext in the merged  
application; context1, context2 - the contexts that are being pulled up in the new  
merged application; App1, App2 - the applications that the linked contexts are  
coming from.  
  
//Remarks: creates a new isoContext.
```

Appendix E – A Summarized Demo Of Merging

Evolution of COIN Applications: From Single Apps to App Merging

Airfare, Car Renter and Merged Travel App



This demo presents the development of two independent **Context Interchange (COIN)** applications: Airfare Aggregator and Car Renter.

Airfare Aggregator brings together data from many independent airfare aggregator websites such as Orbitz, Expedia, Itn (an American Express site), TravelSelect (a UK site), etc, so that they can be queried in a context blind fashion for the lowest fares across all of them.

Car Renter brings together data from many independent US car rental aggregators such as Yahoo Car Rental, Qixo Car Rental, etc so that they can be queried in a context blind fashion for the lowest rates across all of them.

The demo then presents motivation for merging COIN applications and demonstrates the capability of merging and the issues involved through the example of merging the Airfare and Car Renter applications.

The Demos:

- [Airfare Aggregator](#)
- [Car Renter](#)
- [Merged Travel Application](#)

Want to find the lowest airfare?

Do you want to find the lowest price across multiple airfare aggregators such as Orbitz, Expedia, Yahoo, etc?



We will help you find the best fare online.

Where will you fly from? (please enter airport code - eg. NRT for Tokyo, Japan)

Where do you want to fly to? (please enter airport code - eg. BOS for Boston, MA)

When do you want to depart? Aug 2003

When do you want to return? Aug 2003

[What is the motivation for this application?](#)

created by M. Bilal Kaleem - June 2003

Best Prices Found!

The following are the best fares returned by each of the sources for the dates: Aug 25, 03 to Sep 13, 03.

Provider	Departure	Destination	Airline	Price
TravelSelect	NRT	BOS	Delta Airlines 0056	1088.86134
Yahoo	NRT	BOS	 United Airlines	4502.95
Itn	NRT	BOS	American Airlines*	4534.4

Motivation: Multiple Sources means CONTEXT DIFFERENCES



Meet Dora the explorer - quite the world traveler. But there is one place she has not yet been – where East meets West – the city of Istanbul, Turkey, lying both in Asia and Europe. But before she can pack her bags, she must first endure perhaps the most tedious part of traveling: finding the cheapest flight. Being the thrifty traveler that she is, the first thing she does is to check multiple airfare aggregator websites.

Why multiple sources?

One aggregator site may not always have the lowest fare because often have different arrangements with various airlines to give their fares special consideration. Thus:

Different sites may present the same flights but different prices for the same trip

Different sites may present completely different flights and prices for the same trip

For example, Dora sent the **same query** to Orbitz and Travelocity and got a **\$600 difference for their best fares to Istanbul**:



PRICE (USD)	AIRLINE	TIMES	FROM (airport codes)	TO (airport codes)
SELECT \$1234 per person	Delta Air Lines 6238 operated by partner airline Wed, Jun 25	12:05p-1:20p plane change	Boston (BOS)	New York (JFK)
	Delta Air Lines 72 Wed, Jun 25	5:40p-10:35a	New York (JFK)	Istanbul (IST) total duration: 15h 31m



Travelocity

Departure	Arrival	Airline	Travel Time	Price
6:40pm Boston, MA (BOS)	2:15pm Istanbul, Turkey (IST) Next day arrival	KLM Airlines Flight 6038 operated by Northwest Airlines /	1 Stop Change planes in Amsterdam, Netherlands (AMS) Travel Time: 12hrs 35min	Roundtrip Fare USD 1890 <input type="button" value="Select"/>

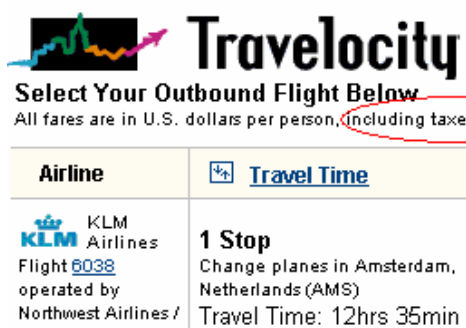
But having to run the same query on multiple sites, juggling between them, trying to remember which offered what fare, while adjusting various dates and times of travel to find the best price is hard.

Also, **different sites means Dora has to interpret differences in the meaning of the results, which is hard. These context differences should be resolved automatically.**


CONTEXT ISSUES

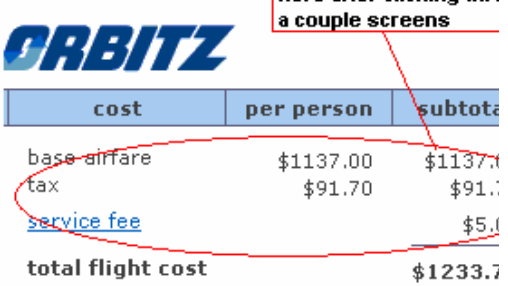
1. What does price include?

Taxes? Service fees? For example, Orbitz includes taxes but does not include a service fee until a screen much later in the process. Expedia includes a service fee right from the beginning. So for each site, Dora would have to account for context differences when comparing the prices.



Travelocity
Select Your Outbound Flight Below
All fares are in U.S. dollars per person, including taxes and fees.

Airline	Travel Time	Price
 KLM Airlines Flight 6038 operated by Northwest Airlines /	1 Stop Change planes in Amsterdam, Netherlands (AMS) Travel Time: 12hrs 35min	Roundtrip From USD 1890 <input type="button" value="Select"/>



ORBITZ

cost	per person	subtotal
base airfare	\$1137.00	\$1137.00
tax	\$91.70	\$91.70
service fee		\$5.00
total flight cost		\$1233.70

2. What about different currencies?

Aggregators are based in many countries and return fares in different currencies. Dora would need to determine what currency the source uses, what the latest exchange rate is and then convert the price quoted.



3. What about paper ticket charges?

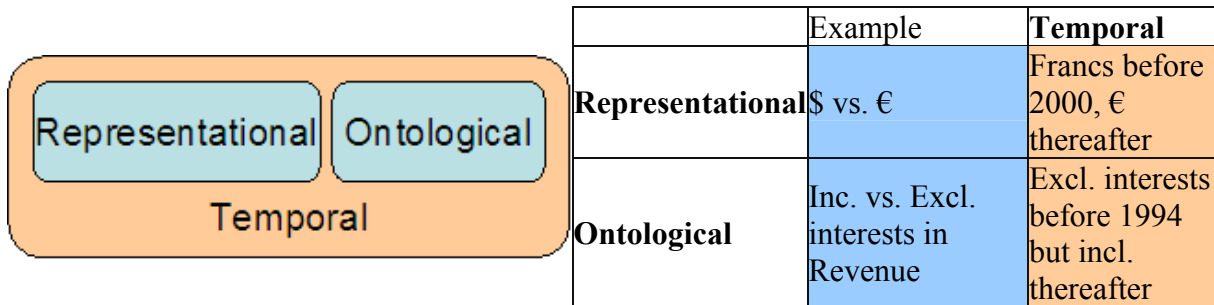
Some flights require travelers to purchase paper tickets or some travelers may even prefer paper tickets. So if Dora wants paper tickets, she would need to determine how much the source (Orbitz, Expedia, etc) charges for paper tickets and add that to her price.

[*Read more about context issues, see examples of other context issues COIN deals with.]

These are just a few of the context issues that might be encountered by Dora. She needs a “**context interchange**” application that can automatically resolve the context differences.

Context Issues Addressed by COIN

We have demonstrated the capability of COIN in solving various context problems, such as differences in unit of measures, scale factors, and inclusion of different items. Here we categorize these issues into three major categories.



1. Representational Contexts

The same attribute can have different presentations, often using different units and scale factors. Current temperature can be reported in Celsius, Fahrenheit, or Kelvin; company revenues can be reported in different currencies using a scale factor of 1, 1000, or one million; product weights can be reported in Newton, pounds, or kilograms. Translations for this category can be easily performed most of the time.

2. Ontological Contexts

A concept may have slight variations, often in the form of varying the components that constitute the concept. Price can be quoted to include or exclude certain items, such as taxes and service charges; profit can include or exclude interests, taxes, and major items. Translations for this category often relies on the existence of functional dependencies between different variations. The new COIN equational solver can solve problems in this category.

3. Temporal Contexts

Anything could change over time. Change of attribute values are dealt using temporal database technologies, which do not address temporal context issues. Temporal contexts refer to the changes of representational and ontological contexts over time. We further categorize temporal contexts to account for some particular issues:

3.1 Change of Identifies

Ideally, identifying attribute should be time invariant. But in practice, key attribute values are often recycled over time: product codes used by manufacturers, course numbers in universities, and stocks symbol in stock exchanges. For example, does "C" stand for CitiGroup or Chrysler?

3.2 Change of Representation (see example in the table)

3.3 Change of Concept

We give an example in the table. In some cases, the components can change even without changing the concept. For example, the concept of Dow Jones Industrial Average is to use an index as the market barometer. This concept has not changed, yet its components have been changed constantly. Among the initial companies, only GE still remains in the Dow today.

3.4 Change of Derivation Methods

Some complex concepts are derived using special methods, which can be improved over time. Major changes in these methods often affect the results and how the results should be interpreted. For example, the change of survey questionnaire for the Gini index; many changes to the accounting rules to national economy.

3.5 Differences in Time as a Data Type

Introducing temporal context will necessarily introduce time as a data type, which can have representational and ontological differences: calendars, granularities, user defined times (e.g., a lecture session) and calendars (MIT academic calendar); commute time (including/excluding time for finding parking).

3.6 Differences in Time Perspective

The same thing can be looked from different time perspectives: 1 million German marks worth a lot 10 years ago, but it worth almost nothing in 1923 when the exchange rate was about 1trillion marks to \$1.

Research is underway to address various temporal issues.

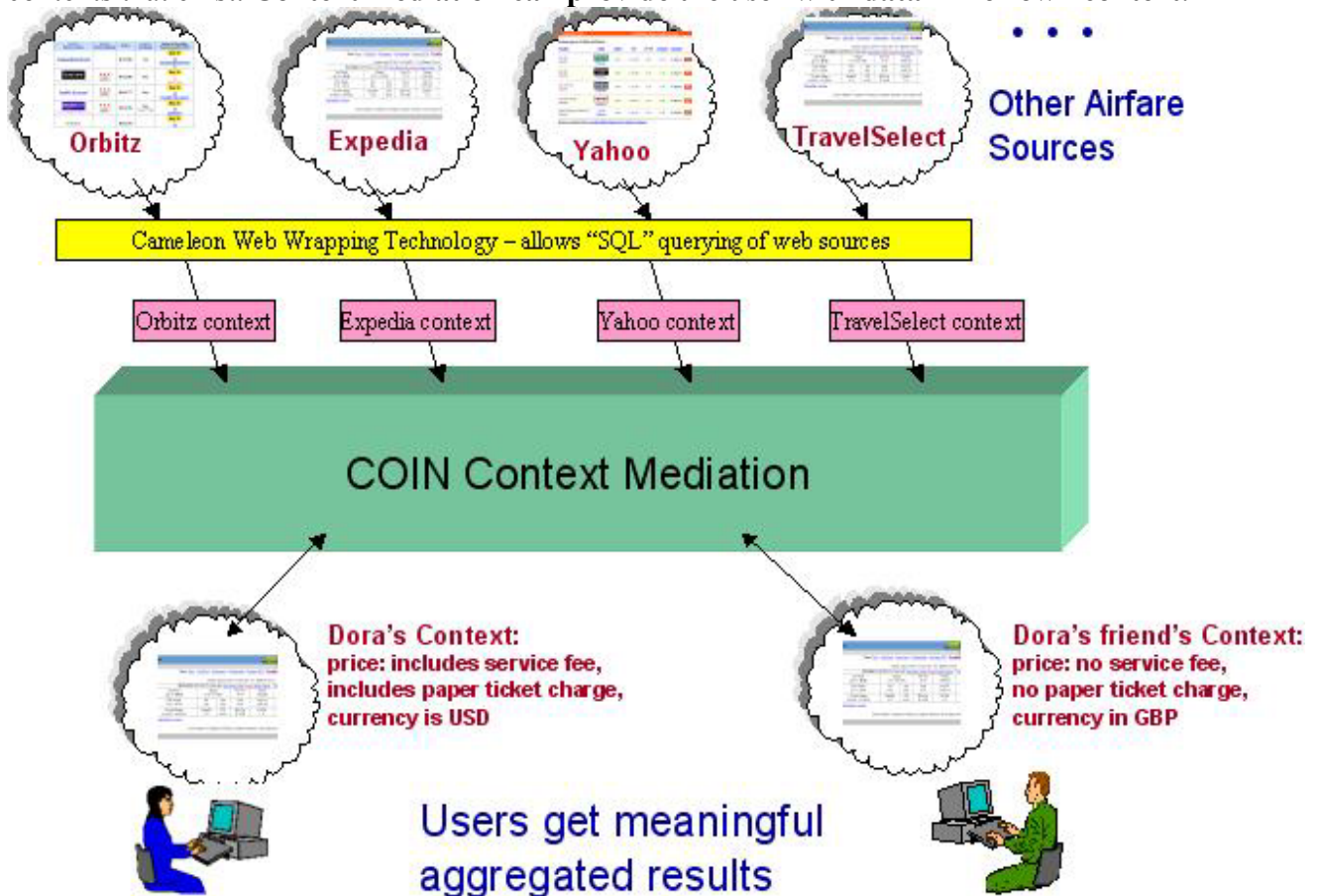
Continue with Demo.

Frustrated by the various context issues, Dora decides to solve the problem presented by the diversity of the meaning of *price* through semantic data integration technologies. **She builds an airfare application using the Context Interchange (COIN) system at MIT.**

Multiple sources and multiple users mean there are different possible interpretations (i.e. different contexts) for the same data.

A "user" or "receiver" context is the set of assumptions a user makes when interpreting/understanding the data.

The COIN system allows users to query application data without worrying about all the contexts that exist. **Context mediation can provide the user with data in her own context.**



How it's done

COIN uses *ontologies* (application domain models) which contain *semantic types*, *modifiers* and *modifier values* to support the notion of context & context mediation.

Entities in the application such as price, airline, departureDate etc, are called *semanticTypes*. *SemanticTypes* that are subject to multiple interpretations (for eg. price) have *modifiers*. It is these *modifiers* that tell the system how to interpret a *semanticType*. Namely, the *modifiers* of a *semanticType* can take on one of many *modifier values* and it is the *modifier value* that tells the system what the *semanticType* means. **For example, the *semanticType*, "price" has a *modifier*, "currency" whose *modifier value* can be either "USD", "GBP", etc, depending on the context.**

COIN provides an application editing tool to build and maintain ontologies, contexts, and to add/remove data sources.

Summary of Steps a query goes thru:

Query and Receiver Context Entry

A user's query is sent to the COIN abduction engine along with the user's context. Next, the abduction engine sees what source the user wants to query and determines the source context.

GCMS Demo for Airfare Aggregator

Receiver context is set to "dora"

Query is looking for fare to Istanbul from traveselect (a UK source)

Conflict Detection

The abduction engine then determines the conflicts (differences in modifier values) between the source context and the receiver context.

Abduction engine has just performed conflict detection

SemanticType	Column	Source	Modifier	Modifier value in source context	Modifier value in target context	Conversion Function
price	IV	traveselect (Name, C2, IST, BOS, 08, 08, 11, 23, Name, C10)	currenc	c traveselect : GBP	dora : USD	olsen_p(V12, V11, V10, V9), value(V12, V8, V7), value(V11, V8, V6), value(V10, V8, V5), currentDate_p(V4), value(V4, V8, V3), value(V9, V8, V3), multiply(V2, V5, V1)

For semanticType price, conflict detected in modifier value for currency: GBP vs. USD

Mediation and Revised Query

The engine then revises the user's query to contain conversion functions that will reconcile the differences between the user's context and the source's context. Note how different (and longer!) the query below is compared to the original user's query in step 1. Without context mediation, the user would have to write queries such as the one below when querying sources with context differences.

Engine revises query to contain conversions that add service fee, paper ticket charge and currency conversion

```

select traveselselect Provider, traveselselect Airline, 'IST', 'BOS',
(((traveselselect.Price+servicefees.ServiceFees)+paper_fees.paperfee)*olsen.rate)
from (select Airline, Price, 'IST', 'BOS', '08', '08', '11', '23', Provider, IsIn
from traveselselect
where Destination='IST'
and Departure='BOS'
and Month1='08'
and Month2='08'
and Day1='11'
and Day2='23') traveselselect,
(select Provider, ServiceFees
from servicefees) servicefees,
(select provider, paperfee
from paper_fees) paper_fees,
(select 'GBP', 'USD', rate, '6/19/03'
from olsen
where exchanged='GBP'
and expressed='USD'
and date='6/19/03') olsen
where traveselselect.Provider = servicefees.Provider
and servicefees.Provider = paper_fees.provider

```

Context definitions for Airfare Aggregator Application:

Context Type	Context Name	includesServiceFee?	includesPaperTktCharge?	Currency
Receiver Contexts	Dora's Friend	No	No	GBP
	Dora	Yes	Yes	USD
Source Contexts	Yahoo	Yes	No	USD
	Expedia	Yes	No	USD
	Orbitz	No	No	USD
	Traveselselect	No	No	GBP
	ltn	No	No	USD

See conversions between the contexts created by M. Bilal Kaleem - June 2003

Conversions between Contexts

Once the ontology and all the context knowledge is specified, the system can take queries in the user's context, retrieve information from various airfare sources, and convert the results into the user's context.

As a reminder, here are the contexts:

<u>Context Type</u>	<u>Context Name</u>	<u>includesServiceFee?</u>	<u>includesPaperTktCharge?</u>	<u>Currency</u>
Receiver Contexts	Dora's Friend	No	No	GBP
	Dora	Yes	Yes	USD
Source Contexts	Yahoo	Yes	No	USD
	Expedia	Yes	No	USD
	Orbitz	No	No	USD
	Travelselect	No	No	GBP
	Itn	No	No	USD

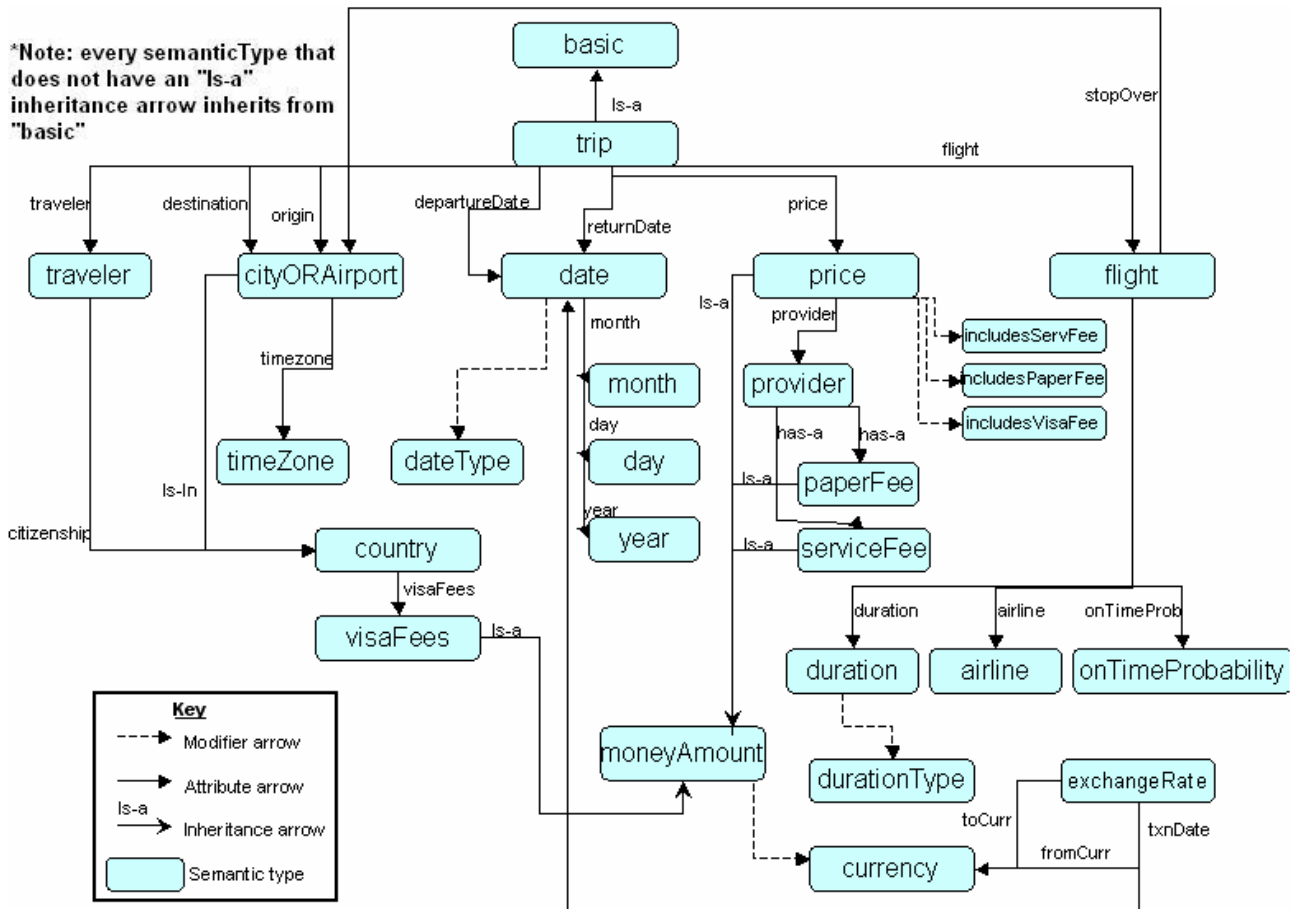
Now Dora wants to go to Istanbul, Turkey, wants a paper ticket, wants the price quoted to include any possible expense (i.e taxes and service fees). She sets the receiver context to "Dora" and here are the conversions the system automatically generates for the data coming from the different sources:

Source	Conversion
Orbitz	No need to add taxes, determine service fee and paper ticket charge for Orbitz and add them
Expedia	No need to add taxes, service fees already included (do nothing), determine paper ticket charge and add it.
Yahoo	No need to add taxes, service fees already included (do nothing), determine paper ticket charge and add it.
Travelselect	No need to add taxes, determine service fee and paper ticket charge for Orbitz and add them, convert everything from GBP to USD.
Itn	No need to add taxes, determine service fee and paper ticket charge for Orbitz and add them

The results from all sources will be merged with context differences reconciled. With this application, finding airfare prices is only a matter of a few mouse clicks. See the domain model (ontology) of Airfare Application. created by M. Bilal Kaleem - June 2003

Airfare Application Ontology

The ontology captures all concepts and their relationships in the application domain. The following figure gives a graphical representation of the airfare application's ontology.



Move on to car rental demo.

created by M. Bilal Kaleem - June 2003

Want to find the lowest car rental price?



Do you want to find the lowest price across multiple car rental aggregators such as Orbitz, Expedia, Yahoo, Qixo etc?

OK, we will help you find the best car rental online

Top of Form

Where do you want to rent a car? (please enter an airport code (eg. "BOS" if you want Boston, MA):

When do you want to rent? Aug 01 2003

When do you want to return the car? Aug 13 2003

Bottom of Form

Bottom of Form

***What is the motivation for this application?**

created by M. Bilal Kaleem - June 2003

Best Prices Found!

The following are the best fares returned by each of the sources for the dates: **Sep 01, 2003 to Sep 13, 2003.**

From Qixo Car Rental Aggregator

Company	Pickup	Price	Rateperiod
Thrifty	BOS	40.99	DAILY
National Car Rental	BOS	43.99	DAILY
Hertz	BOS	68.99	DAILY

From Expedia Car Rental Aggregator

Company	Pickup	Price	Rateperiod
Expedia.com	BOS	200.00	Week
home	BOS	210.00	Week
flights	BOS	211.99	Week
hotels	BOS	220.99	Week
cars	BOS	225.99	Week
vacations	BOS	225.99	Week
cruises	BOS	229.99	Week
deals	BOS	230.99	Week
destinations & interests	BOS	238.99	Week
maps	BOS	238.99	Week
business	BOS	238.99	Week
Dollar Rent A Car	BOS	238.99	Week
Dollar Rent A Car	BOS	240.99	Week
Thrifty Car Rental	BOS	240.99	Week
Thrifty Car Rental	BOS	264.99	Week
Dollar Rent A Car	BOS	264.99	Week
Hertz	BOS	273.99	Week
Thrifty Car Rental	BOS	273.99	Week
Avis	BOS	275.99	Week
Thrifty Car Rental	BOS	298.99	Week
Thrifty Car Rental	BOS	300.99	Week
Dollar Rent A Car	BOS	344.99	Week
Dollar Rent A Car	BOS	369.90	Week

Hertz	BOS	372.99	Week
Avis	BOS	429.99	Week
Hertz	BOS	432.99	Week
Avis	BOS	432.99	Week
Avis	BOS	475.99	Week
Hertz	BOS	200.00	Week
Hertz	BOS	210.00	Week
Avis	BOS	211.99	Week
Avis	BOS	220.99	Week
Thrifty Car Rental	BOS	225.99	Week
Dollar Rent A Car	BOS	225.99	Week
Thrifty Car Rental	BOS	229.99	Week
Hertz	BOS	230.99	Week
Avis	BOS	238.99	Week
Hertz	BOS	238.99	Week
Dollar Rent A Car	BOS	238.99	Week

From Yahoo Car Rental Aggregator

Company	Pickup	Price	Rateperiod
NATIONAL	BOS	33.95	Daily
USAVE AUTO	BOS	39.99	Daily
THRIFTY	BOS	42.99	Daily
HERTZ	BOS	70.99	Daily

Motivation: Multiple Sources mean CONTEXT DIFFERENCES



Joe is Dora's friend. Seeing Dora's efforts, he wanted to build a similar application for himself since he rents cars a lot to travel around the US.

Main Benefit of Car Rental Application: allows users to browse various sources and see prices in a single context.

BUT, there are a number of context issues:

1. What does price include?

Does the price include the **service fee** that an aggregator site such as Yahoo Car rental may charge? Does the price include **taxes**? For each site, the application can automatically figure out what the price includes and account for context differences when comparing the prices.

2. What about date format?

There are differences in the way different sites represent a month. Some use a two-digits notation while others use a three-letter symbol (i.e. **"06"** versus **"Jun"**). The application will resolve that context difference.

3. Airport code versus actual city name?

Sites may be able to understand either city names or airport codes or both (that is, for start and end locations). The application will allow users to enter either city names or airport codes and **resolve the difference in context by automatically converting between city names and airport codes.**

***Read more about context issues, see examples of other context issues COIN deals with. Multiple sources give rise to several context issues. Joe needs a "context interchange" application that can resolve the differences.**

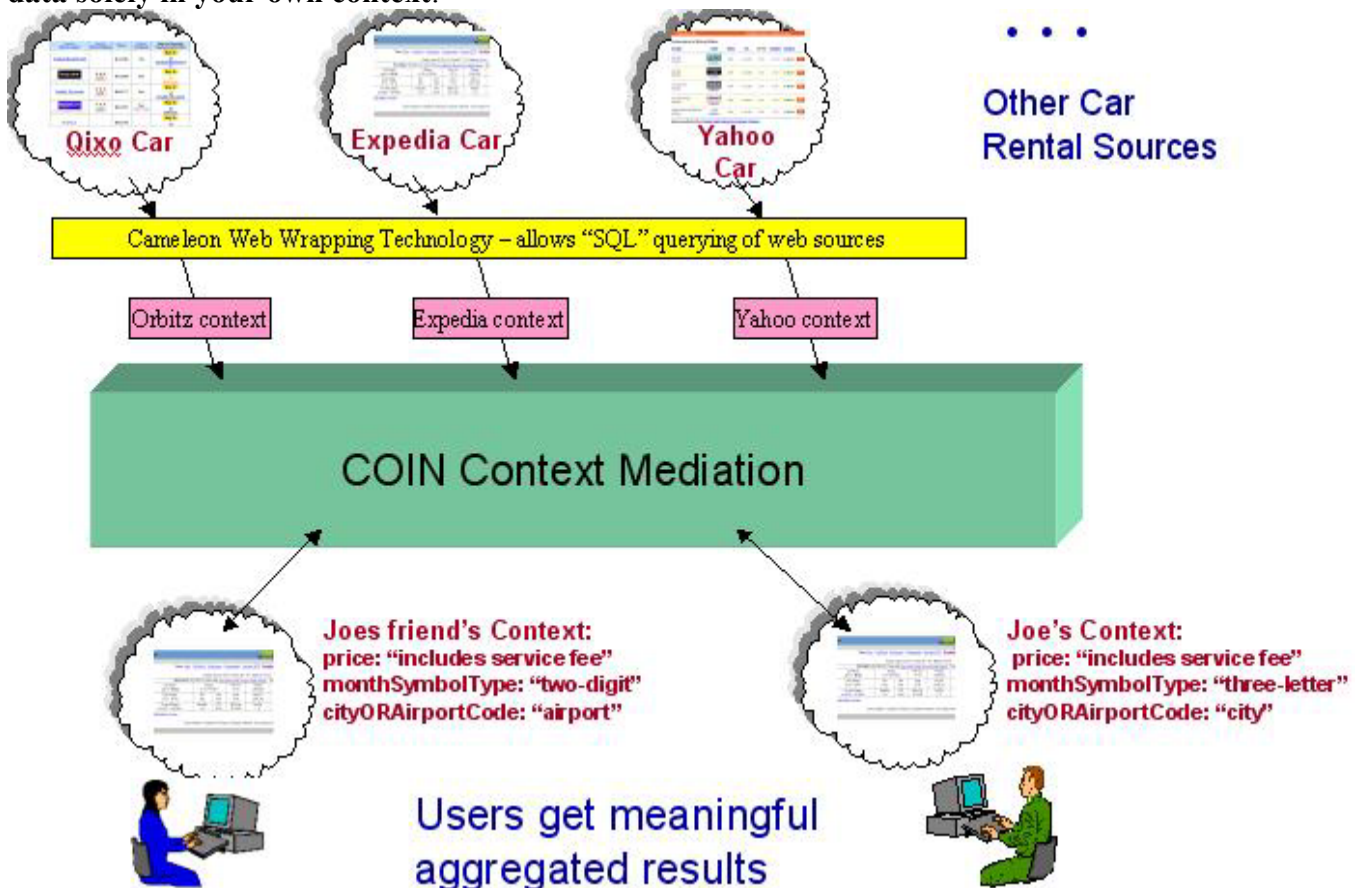
created by M. Bilal Kaleem - June 2003

The COIN Approach

Multiple sources and multiple users mean there are different possible interpretations (i.e. different contexts) for the same data.

A *context* is the set of assumptions one makes in interpreting/understanding the data.

The COIN system allows users to query application data without worrying about all the contexts that exist. COIN does automatic context mediation so that you can deal with data solely in your own context.



How its done

COIN uses *ontologies* (application domain models) which contain *semantic types*, *modifiers* and *modifier values* to support the notion of context & context mediation. Entities in the application such as price, rental, pickupLocation, etc, are called *semanticTypes*. *SemanticTypes* that are subject to multiple interpretations (for eg. price) have *modifiers*. It is these *modifiers* that tell the system how to interpret a *semanticType*. How? Because the *modifiers* of a *semanticType* can take on one of many *modifier values* and it is the *modifier value* that tells the system what the *semanticType* means. For example, the *semanticType*, "month" has a *modifier*, "monthSymbolType" whose *modifier value* can be either "two-digit" or "three-letter" (i.e "06" vs. "Jun") depending on the context.

COIN provides an application editing tool to build and maintain ontologies, contexts, and to add/remove data sources.

Context definitions for Car Rental Application:

Site	includesServFee - modifier of Price	monthSymType - modifier of Month	cityOrAirportCode - modifier of City
yahoo	"Yes"	"three-letter"	"airport"
expedia	"Yes"	"two-digit"	"airport"
qixo	"No"	"two-digit"	"airport"
joe	"Yes"	"three-letter"	"city"
joe's friend	"Yes"	"two-digit"	"airport"

See conversions between the contexts
created by M. Bilal Kaleem - June 2003

Conversions between Contexts

Once the ontology and all the context knowledge is specified, the system can take queries in the user's context, retrieve information from various airfare sources, and convert the results into the user's context.

Site	includesServFee - modifier of Price	monthSymType - modifier of Month	cityOrAirportCode - modifier of City
yahoo	"Yes"	"three-letter"	"airport"
expedia	"Yes"	"three-letter"	"airport"
qixo	"No"	"two-digit"	"airport"
joe	"Yes"	"three-letter"	"city"
joe's friend	"Yes"	"two-digit"	"airport"

Now suppose the user wants to rent a car from San Francisco's airport and the source he wants to query is the Qixo aggregator. He would like the price quoted to include the source's service fee. He sets the receiver context to Dora. Here is the conversion that the system would automatically perform on each result returned from Qixo (conversion from Qixo context to Dora context):

Source	Conversion
Qixo	- convert three-letter month symbols entered by user to two-digit month symbols that source can understand - convert "San Francisco, CA" to "SFO" airport code - lookup serviceFee Qixo charges and add it to price

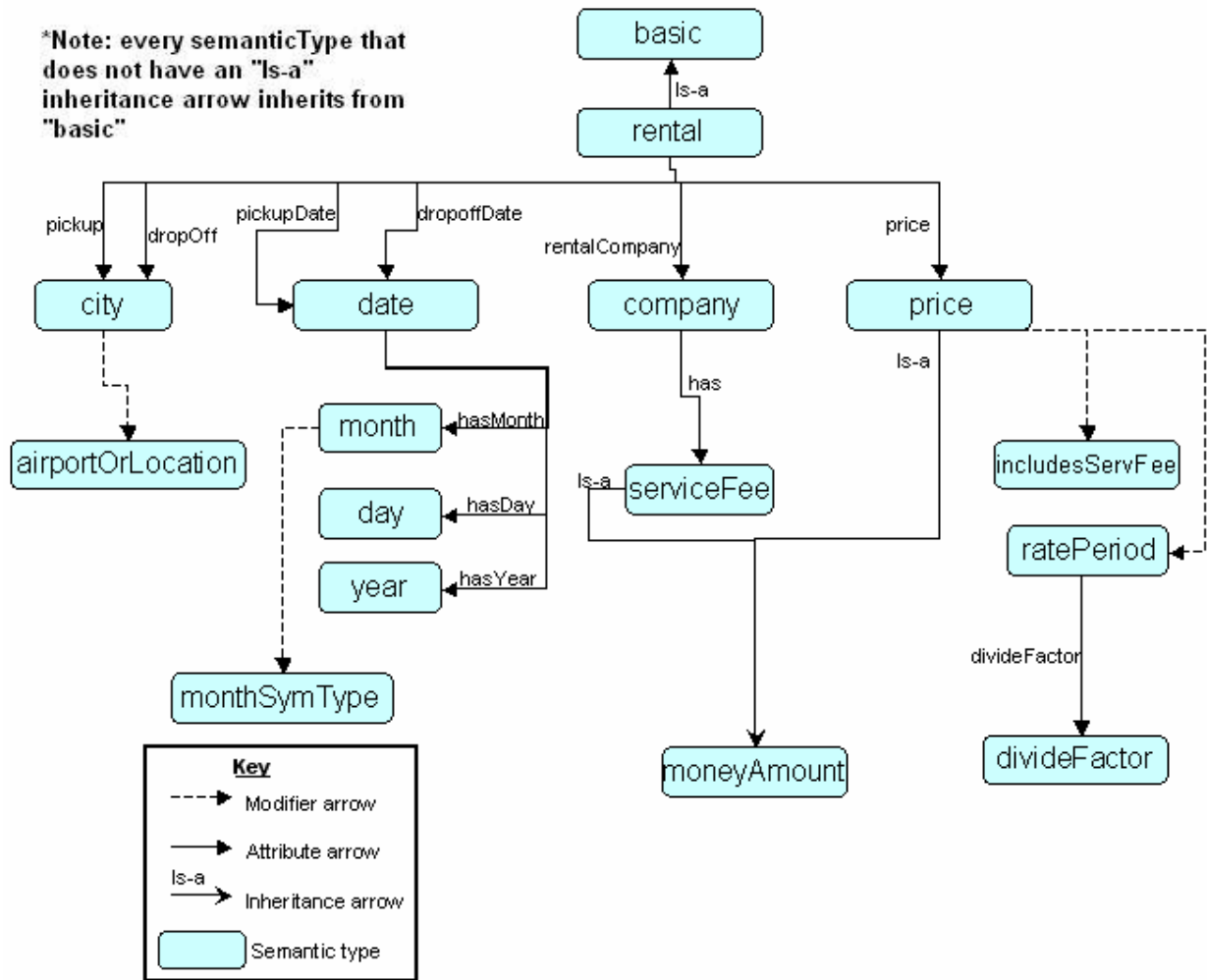
Results from other sources would be similarly reconciled so that user would not have to worry about context differences

See domain model (ontology) of Car Rental Application

created by M. Bilal Kaleem - June 2003

Car Rental Application Ontology

The ontology captures all concepts and their relationships in the application domain. The following figure gives a graphical representation of the ontology for the Car Rental application.



Move onto Merged Travel Application demo

created by M. Bilal Kaleem - June 2003

Merging Airfare and Car Rental - Full Travel Application

Do you want to completely plan your travel needs? Find the cheapest airfare and rent a car at your destination?



We will help you find the best deals online!

Find Airfare

Top of Form

Where will you fly from? (please enter airport code - eg. NRT for Tokyo, Japan)

Where do you want to fly to? (please enter airport code - eg. BOS for Boston, MA)

When do you want to depart?

Aug	▼	01	▼	2003	▼
-----	---	----	---	------	---

When do you want to return?

Aug	▼	13	▼	2003	▼
-----	---	----	---	------	---

Rent A Car There:



Rent a car for duration of stay



Rent a car there only from:

Aug	▼	02	▼	2003	▼
-----	---	----	---	------	---

to:

Aug	▼	12	▼	2003	▼
-----	---	----	---	------	---

Bottom of Form

Bottom of Form










***What is the motivation for merging applications.**

created by M. Bilal Kaleem - June 2003

Best Prices Found!

The following are the best fares and car rentals returned by the sources. Both Airfare and Car Rental are for the dates: **Sep 01, 2003 to Sep 13, 2003**



From Yahoo


yahoo.Destination	yahoo.Airline	yahoo.Price	yahoocar.Company	yahoocar.Price	yahoocar.Rateperiod
BOS	 Northwest Airlines	1482	ENTERPRISE	209.95	Weekly
BOS	 Northwest Airlines	1482	USAVE AUTO	209.99	Weekly
BOS	 Northwest Airlines	1482	DOLLAR	210.00	Weekly
BOS	 Northwest Airlines	1482	ALAMO	219.44	Weekly
BOS	 Northwest Airlines	1482	THRIFTY	220.99	Weekly
BOS	 Northwest Airlines	1482	NATIONAL	235.99	Weekly
BOS	 Northwest Airlines	1482	HERTZ	240.99	Weekly
BOS	 Northwest Airlines	1482	AVIS	240.99	Weekly
BOS	 Northwest Airlines	1482	BUDGET	274.99	Weekly

From Expedia

expedia2.Destination	expedia2.Airline	expedia2.Price	expediacar.Company	expediacar.Price	expediacar.Rateperiod
----------------------	------------------	----------------	--------------------	------------------	-----------------------

From Itn and Qixo-Car

itn.Destination	itn.Airline	itn.Price	qixocar.Company	qixocar.Price	qixocar.Rateperiod
BOS	 UNITED AIRLINES	1878.10	Enterprise	211.95	WEEKLY
BOS	 UNITED AIRLINES	1878.10	Dollar	211.99	WEEKLY

BOS	 UNITED AIRLINES	1878.10	Thrifty	211.99	WEEKLY
BOS	 UNITED AIRLINES	1878.10	Hertz	225.99	WEEKLY
BOS	 UNITED AIRLINES	1878.10	Alamo	236.24	WEEKLY
BOS	 UNITED AIRLINES	1878.10	National Car Rental	236.31	WEEKLY
BOS	 UNITED AIRLINES	1878.10	Budget	274.99	WEEKLY
BOS	 UNITED AIRLINES	1878.10	Avis	304.99	WEEKLY

Motivation: Building larger apps from small apps is realistic and allows re-use



Meet Dora the explorer - quite the world traveler. Previously, Dora had built herself a COIN application that allows her to query multiple online aggregators for the cheapest airfares without worrying about context differences between the sources. She just saw an analogous car rental application. But:

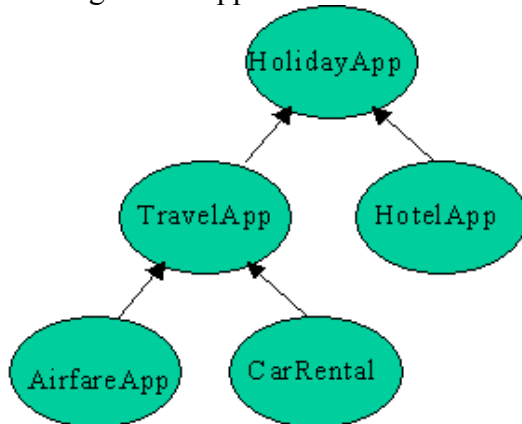
Now that Dora has built her Airfare application and sees that her colleague Joe has built a Car Rental application, she thinks **it would be great to have a general travel application that does both airfare and car rental.**

But she wants to avoid creating a brand new ontology (domain model) that covers everything. She **wants to reuse relevant portions of the existing ontologies.** The solution is COIN application merging.

COIN application merging allows large applications to be built from multiple, small, applications (and their ontologies). This is useful in the real world because designers of applications (and ontologies) rarely have a broad enough vision to predict what will be desired in the future. **It is more realistic to design small, relevant applications and then merge them with other applications as needed in the future.**

No need to create a large new ontology - simply link the "intersecting" portions of the domain models

Merge multiple applications, two at a time, to build larger application. Reuse data, context, ontologies and application code.



COIN merging is driven by context differences - so the merging focuses on differences in the modifiers and related semantic types.

Goals of Merging:

Sources from both applications available seamlessly in one application. Allows you to query sources from both the airfare and the car rental application from one place. So you can find lowest airfare and cheapest car at destination:

Give me the airline and price from expedia and a car rental company and rental rate from yahooCar for the cheapest airfare and car rental for a trip from Boston, MA to San Francisco, CA from Jun 13th 2003 to Jun 19th 2003.

Use context capabilities of one application to benefit the other application:

- **Use currency conversion from Airfare application to benefit car rental** so merged application will support the addition of new, international car rental sources

- **Use city name to airport code conversion from Car Rental application to benefit Airfare**

application so merged application allows airfare queries where user does not need to know airport codes.

Extend the new merged application to add new value beyond just what the two merged applications provide. For example adding a new source or a new context that makes sense in the new merged environment.

- ***add new context called FlyAndRent where price is defined as sum of airfare and car rental price.*** This way, in your query, you simply ask for price and without having to query an airfare source as well as a car rental source (such as in goal #1 above), you are quoted a price that includes both! This context shows that a merged application allows you to extend the two previous applications in ways that you could not have done by simply extending one (or both) of the independent applications.

Doing the merging requires merger axioms.
created by M. Bilal Kaleem - June 2003

How its done - The Merger Axioms

A set of "merger axioms" are needed to link the ontologies of two applications. This **set of merger axioms is relatively small because the majority of the benefit is going to come from** existing ontologies and contexts while the merger axioms are required only to:

- reconcile the differences between the applications
- extend the new merged application

The merger axioms are written in a logic programming language called Prolog. An example of two axioms are as follows:

- rule(merges([appAirfare, appCarRental]), (true)).
- rule(isomodifiertypes(appMergedTravel, appAirFare, price, priceAirfare), (true)).

The first rule declares that the airfare and car rental application have been merged. That one rule causes the merged application to inherit all that exists in both application domains. That rule is sufficient to allow a user to query sources from both applications seamlessly through one query, as if all the sources were from one application.

The second rule would be read as follows in plain English:

The semantic type, *priceAirfare* (from Airfare App) and *price* (from CarRental App) are equivalent because they have the same modifiers, so in the merged application, use *price* to refer to this semantic type.

By declaring the two semantic types to be equivalent with respect to their modifiers, the axiom is saying that any modifier of those two semantic types that may be found in the airfare or car rental application is now to be used in the new, merged application. *price* (which came from car rental) has no notion of currency. But since it has been declared an "isomodifiertype" of *priceAirfare*, it automatically inherits the *currency* modifier (and associated modifier values & conversion functions) of *priceAirfare*. This axiom thus allows sources and prices from the car rental application to leverage currency conversion capabilities of the airfare application.

COIN provides an application merging tool to facilitate merger axiom creation.

(see complete set of merger axioms)

The following is a summary of the merger axioms required for each of the goals of merging mentioned earlier:

The Desired Goal	Merger Axioms Required	Example of Axiom in Prolog (the numbering below corresponds to the numbering in the second column)
Seamless access to sources across both applications	<p>1) Declare that the applications have been merged.</p> <p>All existing sources and contexts will be automatically inherited by the new merged application</p>	<p>1) rule(merges([appAirfare,appCarRental]),(true)).</p>
<p>Use context capabilities of one application to benefit other application.</p> <p>For ex: want <i>price</i> in Car Rental to obtain currency conversion capability from <i>airfarePrice</i> in Airfare</p>	<p>1) Declare that <i>moneyAmount</i> from Car Rental is equivalent to <i>moneyAmount</i> from Airfare. The appropriate modifiers (i.e. currency) and conversion functions will automatically apply.</p> <p>2) For each context that the merged application inherits from Car Rental, declare a modifier value for currency.</p>	<p>1) rule(isomodifiertypes (appMergedTravel, appAirfare, price, airfarePrice), (true)).</p> <p>2) rule(modifier(price, O, currency, expediaCarContext, M), (cste(basic, M, expediaCarContext, "USDollar"))). ... similar rule for the rest of the contexts from Car Rental</p>
<p>Extend merged application with new sources, contexts or modifiers, etc</p> <p>For example, add context FlyAndRent that defines price as (airfare price) + (Car Rental Price)</p>	<p>1) Need axiom for new context</p> <p>2) New axiom for new modifier being added</p> <p>3) Need axioms that give that modifier a value in all of the existing contexts and assign all the existing modifiers a value for the new context</p> <p>4) Need axioms that define conversion functions for the new modifier</p> <p>5) Need axioms that define new attributes used by the new conversion functions that were added</p>	<p>1) rule(contexts([newContextForFlyAndRent]),(true)).</p> <p>2) rule(modifiers(price, [includesCarRental]), (true)).</p> <p>3) rule(modifier(price, O, includesCarRental, doraContext, M), (cste(basic, Modifier, doraContext, "dontIncludeRental"))). ... similar rule for the rest of the contexts in the merged app</p> <p>4) rule(cvt(commutative, price, O, includesCarRental, Ctxt, "dontIncludeRental", Vs, "yesIncludeRental", Vt), (attr(O, month1, M1), plus(airPrice, RentalPrice, Result))).</p> <p>5) rule(attr(Price, month1, Mnth1), (yahoo_p(Price, Mnth1, ...))).</p>

eCOIN Demo for Merged Airfare and Car Rental

Metadata: [[Text Interface](#) | [Ontology](#) | [Context](#) | [Source](#) | [Graphical Viewer](#) | [Internal Representation](#)]
[[Other Demos](#) | [qbe](#)]

Description This example shows an Airfare source (TravelSelect) leveraging the monthSymbol & airportCode conversion capabilities it acquired from Car Rental while still dealing with its own currency & serviceFees/PaperCharge context issues.

SQL

```
select Provider, Airline, Destination, Departure, Price from traveselect
where Destination="Tokyo Japan" and Departure="Boston MA" and
Month1="Aug" and Month2="Aug" and Day1="15" and Day2="30";
```

Context

dora

Stage

- Naive Datalog
- Context Sensitive Datalog
- Conflict Detection
- Mediation
- SQL Translation
- Execution

Result

```
answer('V4', 'V3', "Tokyo Japan", "Boston MA", 'V2'):-
    traveselect('V3', 'V2', "Tokyo Japan", "Boston MA", "Aug",
"Aug", "15", "30", 'V4', 'V1').
```

gcms@mit.edu

eCOIN Demo for Merged Airfare and Car Rental

Metadata: [[Text Interface](#) | [Ontology](#) | [Context](#) | [Source](#) | [Graphical Viewer](#) | [Internal Representation](#)]
[\[Other Demos | qbe\]](#)

Description This example shows an Airfare source (TravelSelect) leveraging the monthSymbol & airportCode conversion capabilities it acquired from Car Rental while still dealing with its own currency & serviceFees/PaperCharge context issues.

SQL

```
select Provider, Airline, Destination, Departure, Price from
travelselect where Destination="Tokyo Japan" and Departure="Boston
MA" and Month1="Aug" and Month2="Aug" and Day1="15" and Day2="30";
```

Context dora

- Stage**
- Naive Datalog
 - Context Sensitive Datalog
 - Conflict Detection
 - Mediation
 - SQL Translation
 - Execution
-

Result

SemanticType	Column	Source	Modifier	Modifier value in source context	Modifier value in target context	Conversion Function
price	IV	travelselect (Name, C2, C4, C4, C6, C6, 15, 30, Name, C10)	currency	c_travelselect : GBP	dora : USD	olsen_p(V12, V11, V10, V9), value(V12, V8, V7), value(V11, V8, V6), value(V10, V8, V5), currentDate_p(V4), value(V4, V8, V3), value(V9, V8, V3), multiply(V2, V5, V1)
price	IV	travelselect (Name, C2, C4, C4, C6, C6, 15, 30, Name, C10)	includesPaperCharge	c_travelselect : no	dora : yes	attr(V7, provider, V6), attr(V6, paperTicketFee, V5), value(V5, V4, V3), plus(V2, V3, V1)
price	C2	travelselect (Name, C2, C4, C4, C6, C6, 15, 30, Name, C10)	includesServFee	c_travelselect : no	dora : yes	attr(V7, provider, V6), attr(V6, serviceFee, V5), value(V5, V4, V3), plus(V2, V3, V1)
month	C6	travelselect (Name, C2, C4, C4, C6, C6, 15, 30, Name, C10)	month2SymbolType	c_travelselect : numeric	dora : threeLetter	month_symbol_converter(V2, V1)
cityORAirport	C4	travelselect (Name, C2, C4, C4, C6, C6, 15, 30, Name, C10)	airportOrLocation	c_travelselect : airport	dora : location	airport_code_lookup(V2, V1)

gcms@mit.edu

eCOIN Demo for Merged Airfare and Car Rental

Metadata: [[Text Interface](#) | [Ontology](#) | [Context](#) | [Source](#) | [Graphical Viewer](#) | [Internal Representation](#)]
[\[Other Demos | qbe\]](#)

Description	This example shows an Airfare source (TravelSelect) leveraging the monthSymbol & airportCode conversion capabilities it acquired from Car Rental while still dealing with its own currency & serviceFees/PaperCharge context issues.
SQL	<pre>select Provider, Airline, Destination, Departure, Price from travelselect where Destination="Tokyo Japan" and Departure="Boston MA" and Month1="Aug" and Month2="Aug" and Day1="15" and Day2="30";</pre>
Context	dora
Stage	<input type="radio"/> Naive Datalog <input type="radio"/> SQL Translation <input type="radio"/> Context Sensitive Datalog <input type="radio"/> Execution <input type="radio"/> Conflict Detection <input checked="" type="radio"/> Mediation <input type="button" value="Submit"/> <input type="button" value="Reset"/>
Result	<pre>answer('V14', 'V13', "Tokyo Japan", "Boston MA", 'V12'):- airport_code_lookup("Tokyo Japan", 'V11'), airport_code_lookup("Boston MA", 'V10'), month_symbol_converter('V9', "Aug"), month_symbol_converter('V8', "Aug"), travelselect('V13', 'V7', 'V11', 'V10', 'V9', 'V8', "15", "30", 'V14', 'V6' 'ServiceFees'('V14', 'V5'), 'V4' is 'V7' + 'V5', paper_fees('V14', 'V3'), 'V2' is 'V4' + 'V3', olsen("GBP", "USD", 'V1', "8/13/03"), 'V12' is 'V2' * 'V1'.</pre>

gcms@mit.edu

eCOIN Demo for Merged Airfare and Car Rental

Metadata: [[Text Interface](#) | [Ontology](#) | [Context](#) | [Source](#) | [Graphical Viewer](#) | [Internal Representation](#)]
[[Other Demos](#) | [qbe](#)]

Description	This example shows an Airfare source (TravelSelect) leveraging the monthSymbol & airportCode conversion capabilities it acquired from Car Rental while still dealing with its own currency & serviceFees/PaperCharge context issues.
SQL	<pre>select Provider, Airline, Destination, Departure, Price from travelselect where Destination="Tokyo Japan" and Departure="Boston MA" and Month1="Aug" and Month2="Aug" and Day1="15" and Day2="30";</pre>
Context	dora
Stage	<input type="radio"/> Naive Datalog <input checked="" type="radio"/> SQL Translation <input type="radio"/> Context Sensitive Datalog <input type="radio"/> Execution <input type="radio"/> Conflict Detection <input type="radio"/> Mediation <input type="button" value="Submit"/> <input type="button" value="Reset"/>
Result	<pre>select travelselect.Provider, travelselect.Airline, 'Tokyo Japan', 'Boston MA', (((travelselect.Price+servicefees.ServiceFee)+paper_fees.paperfee)*olsen.rate) from (select 'Tokyo Japan', airportcode from airport_code_lookup where location='Tokyo Japan') airport_code_lookup, (select 'Boston MA', airportcode from airport_code_lookup where location='Boston MA') airport_code_lookup2, (select mm_number, 'Aug' from month_symbol_converter where symbol='Aug') month_symbol_converter, (select mm_number, 'Aug' from month_symbol_converter where symbol='Aug') month_symbol_converter2, (select Airline, Price, Destination, Departure, Month1, Month2, '15', '30', Provider, IsIn from travelselect where Day1='15' and Day2='30') travelselect, (select Provider, ServiceFee from servicefees) servicefees, (select provider, paperfee from paper_fees) paper_fees, (select 'GBP', 'USD', rate, '8/13/03' from olsen where exchanged='GBP' and expressed='USD' and date='8/13/03') olsen where travelselect.Provider = servicefees.Provider and servicefees.Provider = paper_fees.provider and airport_code_lookup.airportcode = travelselect.Destination and airport_code_lookup2.airportcode = travelselect.Departure and month_symbol_converter.mm_number = travelselect.Month1 and month_symbol_converter2.mm_number = travelselect.Month2</pre>

gcms@mit.edu