CONTEXTUAL ALIGNMENT OF ONTOLOGIES IN THE eCOIN SEMANTIC INTEROPERABILITY FRAMEWORK

Aykut Firat
Northeastern University
Boston, MA USA
a.firat@neu.edu

Stuart Madnick, Benjamin Grosof
Massachusetts Institute of Technology
Cambridge, MA USA
{smadnick, bgrosof}@mit.edu

Working Paper CISL# 2006-01

January 2006

Composite Information Systems Laboratory (CISL)
Sloan School of Management, Room E53-320
Massachusetts Institute of Technology
Cambridge, MA 02142

CONTEXTUAL ALIGNMENT OF ONTOLOGIES IN THE eCOIN SEMANTIC INTEROPERABILITY FRAMEWORK

Abstract

The prospect of combining information from diverse sources for superior decision making is plagued by the challenge of semantic heterogeneity, as data sources often adopt different conventions and interpretations when there is no coordination. An emerging solution in information integration is to develop an ontology as a standard data model for a domain of interest, and then to define the correspondences between the data sources and this common model to eliminate their semantic heterogeneity and produce a single integrated view of the data sources. We first claim that this single integrated view approach is unnecessarily restrictive, and instead offer the view that ontologies can simultaneously accommodate multiple integrated views provided the accompaniment of *contexts*, a set of axioms on the interpretation of data allowing local variations in representation and nuances in meaning, and a *conversion function network* between contexts to reconcile contextual differences. Then, we illustrate how to achieve semantic interoperability between multiple ontology-based applications. During this process, application ontologies are aligned through the reconciliation of their context models, and a new application with a virtual merged ontology is created. We illustrate this alternative approach with the alignment of air travel and car rental domains, an actual example from our prototype implementation.

Keywords: Intelligent Information Integration, Query Rewriting, Ontology Merging

1. INTRODUCTION

The globalization of information on the Internet presents significant opportunities and challenges at the same time. The prospect of combining information from diverse sources for superior decision making is plagued by the challenge of semantic heterogeneity, as data sources often adopt different conventions and interpretations when there is no coordination. For example, on the web, a European site lists airfare in Euros, while a USA-based one lists them in Dollars; the airfare in one contains all the taxes and fees, while in another it does not. Furthermore, users of these web sites have their own assumptions about what the data means, which sometimes do not correspond to the reality of the actual web sites.

There are several efforts focused on addressing this semantic interoperability problem. Probably the largest of these efforts is the "Semantic Web" [Berners-Lee et al. 2001]. An emerging solution in these efforts is to develop an ontology as a standard data model for a domain of interest, and then to define the correspondences between the data sources and this common model to eliminate their semantic heterogeneity ([Rahm and Bernstein 2001], [Halevy et al. 2003]). Furthermore, mappings between similar or complementary ontologies are envisioned to achieve semantic interoperability between multiple domains of interest; thereby indefinitely extending the web of semantically connected data sources [Ives et al. 2004].

There are, however, a number of problems with the above approach. First, ontology developers need to standardize the exact meaning and representation of ontological terms. This requirement turns ontology development and adoption into a standardization process which is notoriously arduous and resource greedy. As a consequence, projects involving ontology development are often long; very often longer than initially planned; and too often delayed *ad eternam*.

Second, developing a standard ontology *eliminates* the semantic heterogeneity and locks the receivers into a single integrated view of the data sources. A more flexible approach would preserve the semantic heterogeneity while *reconciling* the semantic conflicts between sources and receivers, and offer multiple integrated views of the data sources based on receiver choices.

Third, mappings in these approaches are defined between sources and ontologies, or between ontologies and ontologies. This kind of a mapping architecture does not allow a clean modularization and reuse of mappings, thus requires unnecessary large amounts of extra work for defining and managing the evolution of mappings to achieve large scale semantic interoperability.

In this paper, we propose an abstraction to handle semantic reconciliation, in which a context is defined, independent of any data source, and semantic reconciliation is performed at the context level by defining conversion functions between contexts as a network. This alternative approach is realized through the *extended COntext INterchange* (eCOIN) framework, a logic based knowledge representation framework formally defined in Section 2.2. eCOIN assumes the existence of an ontology to tie the sources, but this ontology does not act like the "global schema" of the database integration era [Batini et al 1986]. The ontology acknowledges the *minimal* agreements between the data sources, and is coupled with a well-defined (yet extensible) context model to allow variations in representation and nuances in meaning, thus effectively maintaining multiple integrated views of the data sources. The ontological agreements are minimal in the sense that representational and semantic differences can be deferred to contextual axioms and need not be fixed. For example, the ontological term *airfare* should be acknowledged to refer to the price of an airplane ticket in the most general sense, but the specifics of the reference (e.g. round trip vs. one-way, Dollar vs. Euros, including taxes or not) need not be spelled out in the ontology. The ontology defines the dimensions of the possible specializations (e.g. currency) but leaves the particular choices (e.g. US dollars vs. Euros) to the contexts. This approach dramatically shortens the ontology development process by shifting the focus from the specifics to the generics, and allows gradual incorporation of the specifics into the data model. Furthermore, this abstraction allows us to construct a conversion function network independent of any local data models, thus facilitates modularization and reuse of semantic and representational mappings.

In the following sections, we describe the details of the eCOIN approach with simplified yet illustrative examples. First we discuss the case where a single ontology with an associated context model is used to tie multiple air travel web sites. Then, we consider multiple systems modeled this way (with the addition

of an example from the car rental domain) and discuss an approach to achieve interoperability between them without locking users into a single predefined view. The flexibility of our approach and methodology is illustrated with the alignment of the air travel and car rental domains, an actual example from our prototype implementation.

2. SINGLE ONTOLOGY, MULTIPLE VIEWS

We start this section with a simplified scenario to illustrate the semantic interoperability problem involving a number of heterogeneous airfare data sources and the solution offered by eCOIN. In this scenario and others, eCOIN system acts as a middleware accepting naïve user queries and rewriting them into mediated queries using the shared ontology or ontologies, contexts of the data sources and receivers (users), and the conversion function networks as shown in Figure 1. When the mediated queries are run against the data sources, the results are in the form the users[1] expect, because semantic conflicts are addressed with appropriate embedded conversions.

The airfare data sources in our first scenario are semi-structured web sites, but can be treated as structured data sources by using the Cameleon web wrapper engine [Firat et al. 2000]. Throughout this paper, we limit the scope of the semantic interoperability problem to querying multiple semantically heterogeneous data sources. We use the relational model in describing the data sources, and the widespread query language SQL for formulating the user queries. Our approach, however, offers a general logic-based framework and has wider applicability.

Furthermore, we assume that the queries are not expressed against a "mediated ontology", but directly against the data source schemas. When this assumption becomes impractical, the queries may be seen as the outcome of a pre-processing step, in which a query against a mediated ontology has been rewritten against the underlying data sources by using one of the standard techniques such as *answering queries using views* [Halevy 00]. This pre-processing step, in our case, need not be concerned with the semantic

[1] There are typically two types of "users": A "developer" user that writes the actual software that issues SQL requests to the databases and provides a "user friendly" interface (usually via a web browser) so that "end" users can make their requests using simple pull-down menus and other such means. In both cases, these users need not know the actual semantics of the sources. Although the examples in this paper show the use of SQL, the key issue of mapping source contexts to receiver contexts applies to both types of users.

conflicts between the data sources (and users), thus it must be clear that our approach complements rather than competes with such approaches.

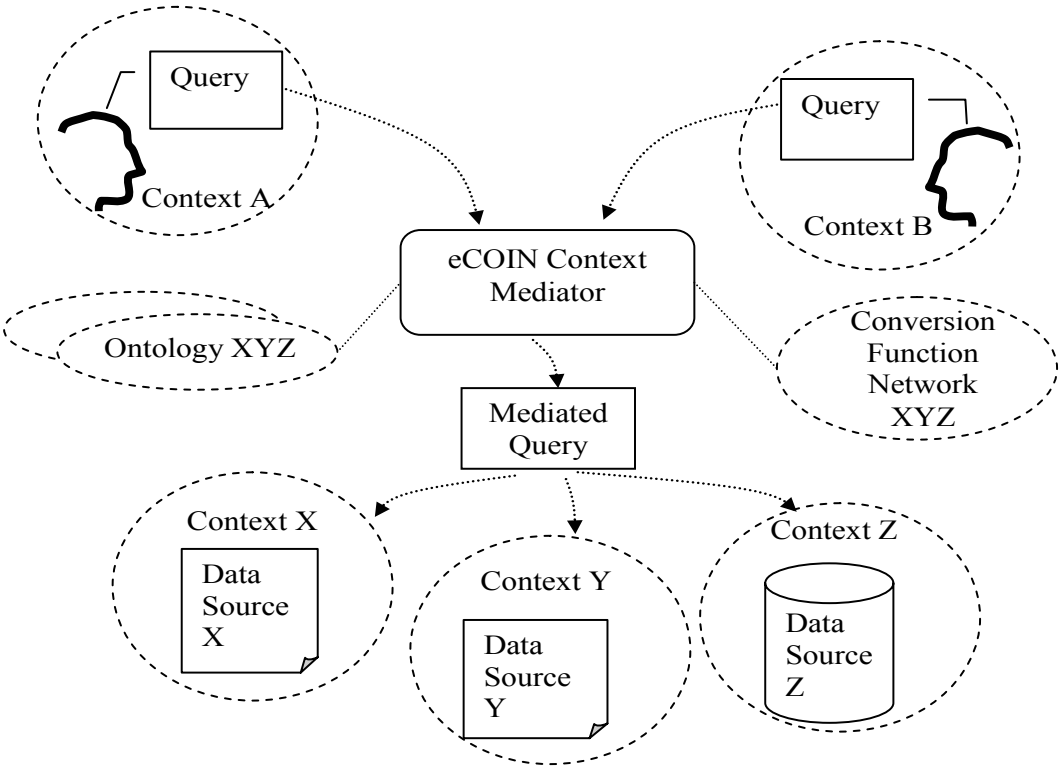


Figure 1 Context Mediation
eCOIN mediates user queries by using the ontology of data sources, contexts of users and data sources, and the conversion function network between contexts.

2.1 MOTIVATIONAL EXAMPLE

Under the air travel scenario, shown in Figure 2, the user wants to query the *cheaptickets* and *eurotickets* web sites through a relational wrapper interface, but she does not have the time, energy, and resources to understand what the data actually means in these data sources. The assumptions (i.e., context) of the user are shown at the top left of Figure 2 [2]. For example, the user is expecting the answer to be in Turkish Liras. Naively, the user formulates the following query hoping to obtain the bottom-line prices of flights from Boston to Istanbul departing on June 1st and returning on July 1st from both sources sorted in ascending order.

[2] Similarly, the contexts of the two sources are shown in the center of Figure 2.

AIRFARE

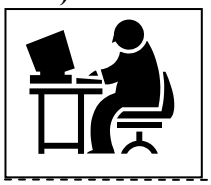
User A in Context C_UA

* Fares are expected to be bottom-line price (round trip, includes taxes and fees)

* Departure and Destination locations are expressed as city names

* Currency is **Turkish Liras (TLR)**

* Today's date: **05/01/05**



Q1: (SELECT Airline, Price FROM CheapTickets
WHERE DepDate = "06/01/05"
and ArrDate= "07/01/05" and
DepCity= "Boston"and ArrCity= "Istanbul"
UNION
SELECT Airline, Price FROM EuroTickets
WHERE DepDate = "06/01/05"
and ArrDate= "07/01/05" and
DepCity= "Boston"and ArrCity= "Istanbul")
ORDER BY ASC;

CheapTickets in Context C_CT

* All fares are for each way of travel and do not include fees and taxes.
* Service fee of $5 is charged
* Departure and Destination locations are expressed as three letter airport codes
* Currency is USD
* Lufthansa offers 10% discount if the airfare is bundled with National car rental

cheaptickets

ID (I)	Airline (A)	Price (P)	Tax (Tx)	DepDate (DD)	ArrDate (AD)	DepCity (DC)	ArrCity (AC)
1	British Airways	495	75	06/01/05	08/01/05	BOS	IST
2	Lufthansa	510	77	06/01/05	08/01/05	BOS	IST
…	…	…	…	…	…	…	…

EuroTickets in Context C_ET

* All fares are round trip and include all fees (service fee of $5) and taxes.
* Departure and Destination locations are expressed as three letter airport codes
* Currency is USD

ID (I)	Airline (A)	Price (P)	Tax (Tx)	DepDate (DD)	ArrDate (AD)	DepCity (DC)	ArrCity (AC)
1	Delta Airlines	1200	160	06/01/05	08/01/05	BOS	IST
2	United Airlines	1145	135	06/01/05	08/01/05	BOS	IST
…	…	…	…	…	…	…	…

Ancillary Sources

currencyrates

FromCur	ToCur	eRate	Date
TLR	USD	0.75	05/10/05
USD	TLR	1.33	05/10/05
USD	EUR	0.80	05/10/05

cityairport

City	Airport
Boston	BOS
Istanbul	IST
…	…

Figure 2 Airfare Example Scenario

Q1:

```
(SELECT Airline, Price FROM cheaptickets WHERE DepDate = "06/01/05"
and ArrDate =  "07/01/05" and DepCity = "Boston" and ArrCity = "Istanbul"
UNION
SELECT Airline, Price FROM eurotickets WHERE DepDate = "06/01/05"
and ArrDate =  "07/01/05" and DepCity = "Boston" and ArrCity = "Istanbul")
ORDER BY PRICE ASC;
```

Because the data sources use airport codes to identify the destination and departure locations, this query returns empty results. Suppose that the user, somehow, figures out this conflict and reformulates the query Q1 by replacing the city names with airport codes (call this new query Q1′). This time the following data are returned from the data sources:

Airline	**Price**
British Airways	495
Lufthansa	510
United Airlines	1145
Delta Airlines	1200

Table 1. Result of query Q1′

These results, however, confuse the user even more, because the user is not aware of the semantic conflicts summarized in Table 2 below, thus cannot make an informed decision. Table 2 shows that the user has conflicts on currency and airport identifier format with both sources, and on what is included and covered (one-way or round trip) in the price with the cheaptickets source.

	Money Amounts	**Price**		**Airport Identifier**
Conflict dimension →	*Currency*	*Inclusion*	*Coverage*	*format*
User	**TLR**	**taxes+fees**	**Round-trip**	**City name**
Cheaptickets	USD	nominal	One-way	Airport Code
eurotickets	USD	**taxes+fees**	**Round-trip**	Airport Code

Table 2 Semantic Conflict Table for the User and the Data Sources

If the original query Q1 were submitted to the eCOIN system, Q1 would be rewritten into the following mediated query MQ1, which, when executed, translates the source data into the form and meaning the user expects. As seen below, the translation is accomplished with the help of arithmetic expressions, and the use of external sources embedded in conversion functions (shown as ancillary sources in Figure 2) that perform the currency and airport identifier conversions.

MQ1: (SELECT Airline, (2* (Price+Tax) + 5) * eRate
FROM cheaptickets, currencyrates, (select Airport from cityairport where city= "Boston") cityairport1, (select Airport from cityairport where city= "Istanbul") cityairport2
WHERE DepDate = "06/01/05" and ArrDate="07/01/05" and DepCity= cityairport1.Airport and ArrCity= cityairport2.Airport and fromCur= "USD" and toCur= "TLR" and Date= "05/10/05"
UNION
SELECT Airline, Price * eRate
FROM eurotickets, currencyrates, (select Airport from cityairport where city= "Boston") cityairport1, (select Airport from cityairport where city= "Istanbul") cityairport2
WHERE DepDate = "06/01/05" and ArrDate="07/01/05" and DepCity= cityairport1.Airport and ArrCity= cityairport2.Airport and fromCur= "USD" and toCur= "TLR" and Date= "05/10/05")
ORDER BY ASC

When query MQ1 is executed, the following results, which reflect the bottom-line prices in Turkish Liras, would be obtained and the user can now make a better decision.

Airline	Price
United Airlines	1527
British Airways	1527
Lufthansa	1572
Delta Airlines	1600

Table 3 Result of MQ1

2.2 REPRESENTATION OF THE MOTIVATIONAL EXAMPLE IN eCOIN

The semantic interoperability described in the previous section is accomplished with a number of declarative statements using the eCOIN semantic interoperability framework, which is a generic logic-based data model that provides a template for the integration of heterogeneous data sources and built on top of and significantly extends the earlier works of [Siegel and Madnick 1991], [Sciore et . al 1994] and [Goh 1996]. This template is defined as follows:

Definition: (eCOIN Framework)

An eCOIN framework is a quadruple **(O, S, C, M)** where each component is a set of logical predicates. **O** corresponds to *ontology* that includes both the *domain and context model*; **S** corresponds to *source declarations*; **C** corresponds to *context (instances)*; and **M** corresponds to *mappings* (in the form of a *conversion function network*) defined between contexts.

In this framework, sources **(S)** and contexts **(C)** are described with respect to the ontology **(O).** Mappings **(M)** are structured according to the context model to enable translation between different contexts. Below each component is described in detail.

ONTOLOGY

Ontology in eCOIN includes both the domain and context model. As in other data integration frameworks, an eCOIN domain model is used to define a common type system for the application domain (e.g., financial analysis, travel information) corresponding to the data sources that are to be integrated. Because the scope of the problem has been limited to querying semantically heterogeneous data sources, the current conceptualization of an ontology is kept as simple as possible. Like many other conceptual models, an eCOIN domain model consists of a collection of (object) *types,* which may be related in a subtype hierarchy. Types have *attributes* to represent both the individual properties of objects and relationships between objects (both things and their properties are uniformly represented as objects).

The types in an eCOIN domain model are *semantic types*, in that they represent the *generic* semantics of the concepts used in the various data sources. A semantic type is impartial to the exact representation or meaning of its instances in specific contexts and encapsulates all. The various specializations of these concepts used by different sources or receivers are described using a special kind of property called a *modifier*. The modifiers in an ontology are chosen to explicitly describe the contextual specializations of the generic types used by the sources and receivers. For example, as shown in Figure 3, the generic ontological term *airfare* represented by the large cube can be

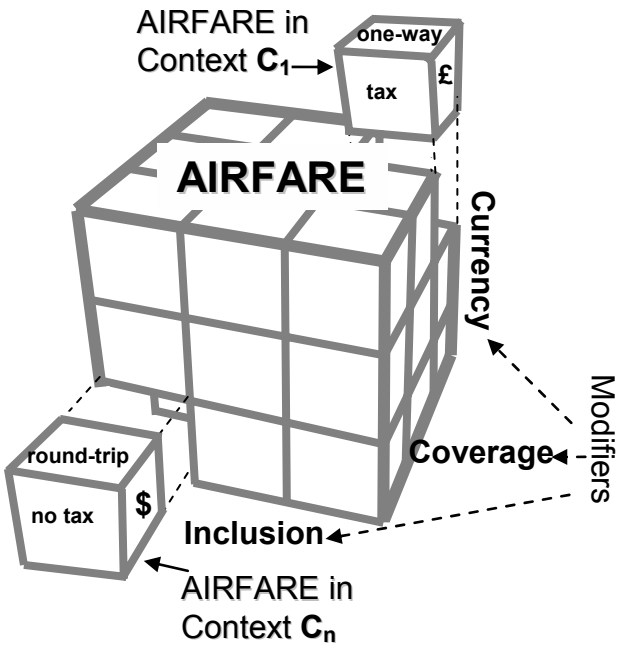


Figure 3 Multi-dimensional modification of the ontological term airfare

specialized along three modification dimensions of *{Coverage, Currency, Inclusion}* (refer to Table 2). Different values of these modifiers identify the different component cubes of the overall airfare cube that may be adopted by different sources and receivers.

The *modifiers* in an ontology collectively define its *context model*; and the collection of *modifier objects* that describe the specializations that can be made by a given source or receiver defines its *context*. Context declarations are source independent, thus multiple sources or receivers may use the same context (use the same specializations for various values), but often different sources use different contexts.

Modifiers themselves are semantic types, thus can be subject to specialization (e.g. There are multiple ways to represent currency, such as USD vs. $.) This can be handled via defining modifiers of modifiers. In Figure 4, this situation is illustrated by a *CurrencyFormat* modifier for the *Currency* modifier. For objects without modifiers, the context model implies a *current* existence of a common representation and meaning across the sources and receivers. If this assumption changes at a later time, new modifiers can be introduced, further slicing and dicing the generic concepts.

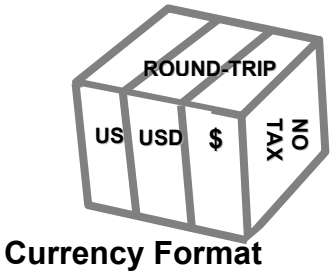


Figure 4 Modifiers can have modifiers

In Figure 5, we illustrate the ontology, context instances and the conversion function network that corresponds to our motivational airfare example. To make our description more concrete, we also provide below the logical predicates used in eCOIN to represent the domain and context model: (The omitted - trivially predictable- predicates are indicated with three dots.)

Domain Model

Types:

```
semanticType(ticket).semanticType(airport).….semanticType(coverageType).
```

Type hierarchy:

```
isa(airfare, moneyAmt). isa(tax, moneyAmt).
```

Attributes/Relationships:

```
serviceFee(ticket,moneyAmt).….hasTax(ticket, tax).
```

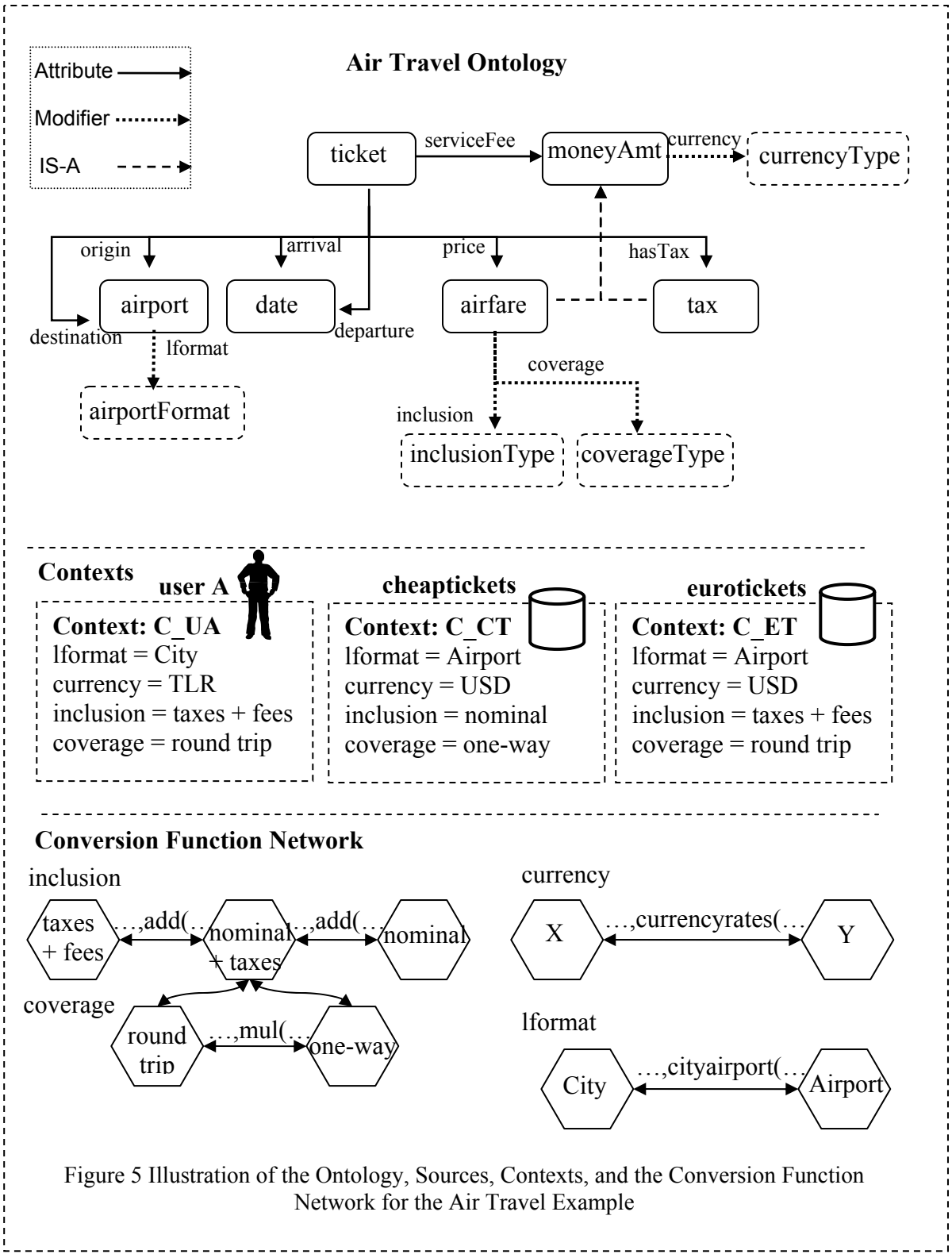


Figure 5 Illustration of the Ontology, Sources, Contexts, and the Conversion Function Network for the Air Travel Example

Context Model

Modifiers:

```
lformat(airport, Context, airportFormat).
inclusion(airfare, Context, inclusionType).
coverage(airfare, Context, coverageType).
currency(moneyAmt, Context, currencyType).
```

The variable 'Context' in the Context Model signifies that a modifier is defined with respect to a given context, thus may acquire different values in different contexts.

SOURCES

Sources in the eCOIN framework are uniformly treated as relational sources (i.e., as having relational schemas). Many non-relational sources, such as HTML and XML web sites and web services, can be transformed into relational sources via wrappers [Firat at al. 2000]. A wrapped web source, for example, can be represented in logical predicates as:

```
cheaptickets(I, A, P, Tx, DD, AD, DC, AC)
```
[3]

In the eCOIN framework, these are called *primitive relations*, because these sources are not yet tied to an ontology. These primitive relations are elevated into *semantic relations* by annotating the semantic type and context of each primitive relation.

The semantic relation cheaptickets' can then be expressed as follows[4]:

```
cheaptickets'(I',A',P',Tx',DD',AD',DC',AC')←
I'=object(ticket,I,c_ct,cheaptickets(I,A,P,Tx,DD,AD,DC,AC)),…,AC'= object(…).
```

With this elevation each column of the cheaptickets relation is tied to the air travel ontology. For the `Id` column, for instance, this is accomplished by associating the value `I` (a generic value) with the ticket semantic type in the cheaptickets context `c_ct`. `I'` in the above declaration is the semantic object corresponding to the primitive object (value) `Id` from the cheaptickets relation.

[3] The abbreviations, such as I and A, correspond to the attributes shown in Figure 2.

[4] Notation: We add a single quote ' to semantic objects/relations to distinguish them from primitive ones.

In addition, the attribute relationships defined by the ontology are instantiated as part of source declarations. For example, the `hasTax` relationship would be declared for this source as follows:

```
hasTax(I',Tx')←cheaptickets'(I',_,_,Tx',_,_,_,_).
```
[5]

This declaration means that the `tax` of a semantic object `I'` is another semantic object `Tx'`, both of which can be obtained from the semantic relation `cheaptickets'`.

CONTEXTS

For *sources*, contexts define the specializations used for the underlying data values; and for *receivers* contexts describe the specializations assumed in viewing the data values. These specializations may be about the representation of data (e.g. European vs. American style date formats) or nuances in meaning (e.g. nominal vs. bottom-line prices). To define a source or receiver context, modifier assignments need to be made. For example, the context labeled as c_ct can be described with the following predicates:

```
currency(MoneyAmt',c_ct,Currency')  ←
  Currency' = object(currencyType,"USD",c_ct,constant("USD")).
inclusion(Airfare', c_ct, Inclusion')←
  Inclusion' = object(inclusionType, "nominal", c_ct, constant("nominal")).
coverage(Airfare', c_ct, Coverage')←
  Coverage' = object(coverageType, "oneway", c_ ct, constant("oneway")).
lformat(Airport', c_ct, LFormat')←
  LFormat' = object(airportFormat, "airport", c_ct, constant("airport")).
```

These modifier declarations, which can include attribute relationships, semantic relations, and some other constructs, explicitly specify which view of the ontology is adopted by the cheaptickets source. Accordingly, the ontology corresponding to the cheaptickets source treats *airfare* as *the one-way nominal price of a ticket in US dollars*. The arrival and departure locations are expressed as airport codes, and money amounts (moneyAmt) in general are in US dollars. These declarations need to be made for all contexts in a similar fashion.

[5] Underscores, as in Prolog, are used to designate *any* value.

MAPPINGS (CONVERSION FUNCTION NETWORK)

Mappings in eCOIN ensure that a view of the ontology adopted in a context is appropriately mapped to a corresponding ontological view in another context. This is accomplished by defining a conversion function network for each ontological term. Conversion functions are *atomically* defined for each modifier dimension as illustrated in Figure 6. As an example, the conversion function for the currency modifier dimension is encoded declaratively in terms of logical predicates as follows:

f_{currency}(X,VS,SC,VCurS,VCurT, TC, VT) ←

```
  value(Today,SC,VToday),systemDate(VToday),value(CurS,SC,VCurS),
  value(CurT,SC,VCurT),   currencyrates'(CurS,CurT,Today,Rate),
  value(Rate,SC,VRate),mul(VS,VRate,VT).
```

For semantic airfare objects, this function uses the modifier value `VCurS` in source context `SC`, and modifier value `VCurT` in target context `TC` to translate the source value `VS` of semantic object `X` to value `VT` in target context. The `value(A,C,B)` predicate used above is read as "the value of semantic object A in context C is B". The function is also using another semantic relation `currencyrates'`; a system function `systemDate(VToday)` and an arithmetic predicate `mul` to express multiplication.

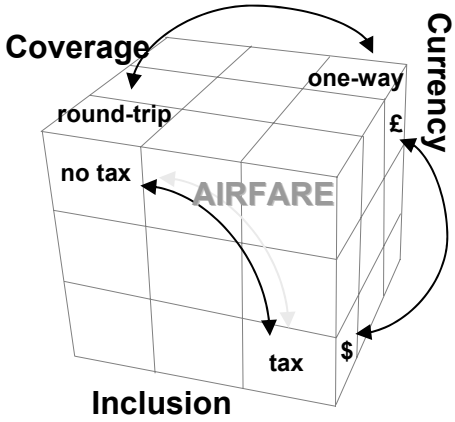


Figure 6 Organization of Conversion Functions for the Ontological term Airfare

As in the currency conversion function example above, conversion functions can sometimes be defined parametrically, thus may cover all of the modifier value pairs with a single function. Within eCOIN conversion functions can be defined as a network to minimize the number of declarations, leaving the tasks of combining, inverting, and simplifying to the mediator. Furthermore, most conversion functions are orthogonal, i.e. they can be applied in any order. When they are not orthogonal, priorities can be assigned to determine the order they are to be executed. The details

of conversion function network organization and mediation techniques used in eCOIN can be found in [Firat 2003].

3. INTEROPERABILITY BETWEEN MULTIPLE eCOIN APPLICATIONS

In the previous section, we have described how a single ontology can be used to achieve interoperability between multiple semantically heterogeneous sources and receivers not by locking them into a single integrated view, but by allowing them to operate within their own context. Many applications using such ontologies describing similar or complementary domains are likely to exist, and achieving semantic interoperability between sources and receivers tied to different ontologies becomes an issue to be addressed. In this section, we first introduce a second application in a related domain, car rental, and then proceed to describe how we achieve interoperability between airfare and car rental applications through contextual alignment of their ontologies.

3.1 AIRFARE WITH CAR RENTAL SCENARIO

Consider now the European car rental application, illustrated in Figure 7. User B in context C_UB poses a query Q2, to find the companies and their prices for car rental to be picked up and dropped off in Istanbul between the dates June 2nd and July 1st. Similar to the airfare scenario, the user and the source have semantic conflicts concerning what is included in the price, and the period of the rental rate (daily vs. total). Unlike the airfare application, however, there is a shared understanding that the currency is Euros, the dates are expressed in European styles, and rental locations are expressed as airport codes; therefore modifiers were not used during the ontology design for date, airport, and the monetary amounts (like price, tax, and fees). Although some of these shared understandings could have been relaxed (through the declaration of appropriate modifiers) with the expectation of future heterogeneities, some of them would likely not be noticed until a heterogeneous source or receiver joins the system. We will see such a situation when we try to use the airfare and car rental applications together, when some of the shared understandings within individual domains will fail to persist in the combined domain.

CAR RENTAL

User B in Context C_UB

* Rentals are expected to be bottom-line price (includes taxes, and fees)
* Rates are for the rental duration
* Currency is Euros

Q2: SELECT Company, Price
FROM cheaprentals
WHERE Class= "Economy" and
PickDate = "02/06/05" and
DropDate= "01/07/05" and
Pickup= "IST" and DropOff= "IST";

cheaprentals in Context C_CR

* Rentals do not include fees and taxes.
* Rates are daily
* National offers 10% discount if the car rental is bundled with a Lufthansa airfare
* Airport concession recovery fee %10
* Sales tax is 5%
* Currency is Euros

cheaprentals

ID (I)	Company (A)	PickUp (PU)	DropOff (DO)	PickDate (PD)	DropDate (DD)	Price (P)	Class (C)	Rate Period (RP)
1	Hertz	IST	IST	02/06/05	01/07/05	23.99	Economy	Daily
2	National	IST	IST	02/06/05	01/07/05	28.79	Economy	Daily
…	…	…	…	…	…	…	…	

Figure 7 Car Rental Example Scenario

Under this scenario, the user query Q2, is submitted to the eCOIN system built with the ontology, context instances, and the mappings shown in Figure 8. This naïve user query, expressed in the context of user B (C_UB), is rewritten into the following mediated query MQ2:

MQ2:
SELECT Company, Price * 34.65
FROM cheaprentals
WHERE Class= "Economy" and PickDate = "02/06/04"
and DropDate= "01/07/04" and Pickup= "IST"and DropOff= "IST"

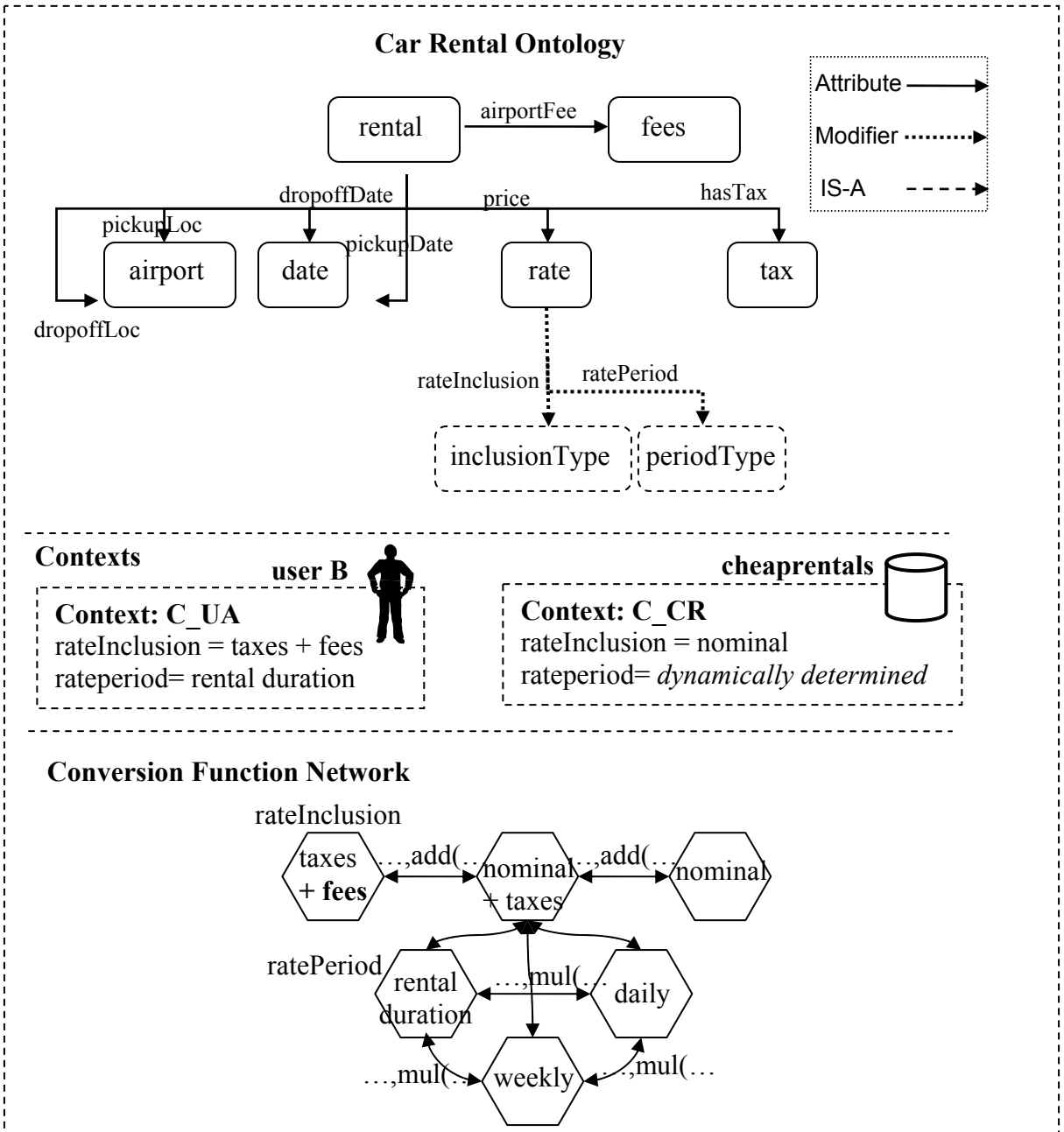


Figure 8 Ontology, Contexts, and the Conversion Function Network for the Car Rental Example

In MQ2, the daily rates given by the sources are converted into the bottom line price requested by the user by multiplying the price by total rental days, the airport concession fee and sales tax ratios (30 * 1.1 * 1.05=34.65). When this query is executed the results shown in Table are returned.

Company	Price
Hertz	831
National	998

Table 4 Result of MQ2

Consider now a third user as shown in Figure 9 who wants to query both the airfare and car rental sources together. The user customarily expresses dates in American style, and locations as city names. With query Q3, she wants to see bottom line prices in Euros including any bundling discounts (refer to Figure 2 & 7 to see the bundling discounts).

AIRFARE & CAR RENTAL

User Merger in Context C_UM

* Both Rentals and Fares are expected to be bottom-line price (including bundling discounts)
* Date is expressed in American style
* Both Rental and flight locations are expressed as city names
* Currency is Euros

Q3:

```
(SELECT "cheaptickets" as fareprovider, Airline, Company, t.Price + r.Price as Total
FROM cheaptickets t, cheaprentals r
WHERE DepDate="06/01/05" and ArrDate="07/01/05" and DepCity= "Boston" and ArrCity= "Istanbul" and
Pickup="Istanbul" and Dropoff="Istanbul" and
PickDate="06/02/05" and DropDate="07/01/05"
UNION
SELECT "eurotickets" as fareprovider, Airline, Company, t.Price + r.Price as Total
FROM eurotickets t, cheaprentals r
WHERE DepDate="06/01/05" and ArrDate="07/01/05" and DepCity= "Boston" and ArrCity= "Istanbul" and
Pickup="Istanbul" and Dropoff="Istanbul" and
PickDate="06/02/05" and DropDate="07/01/05")
ORDER BY Total DESC
```

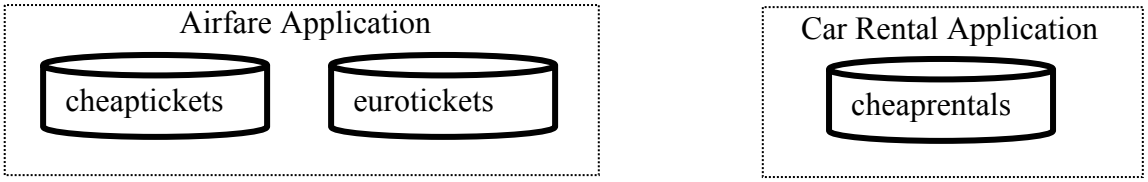


Figure 9. Combined Airfare & Car Rental Example Scenario

Because the airfare and car rental applications have been built independently, a query involving sources from both domains can not be mediated before aligning the individual eCOIN applications. For this purpose a *virtual merger application* from two child applications needs to be built as shown in Figure 10. This application is called virtual, because it doesn't physically contain the merged applications. Instead,

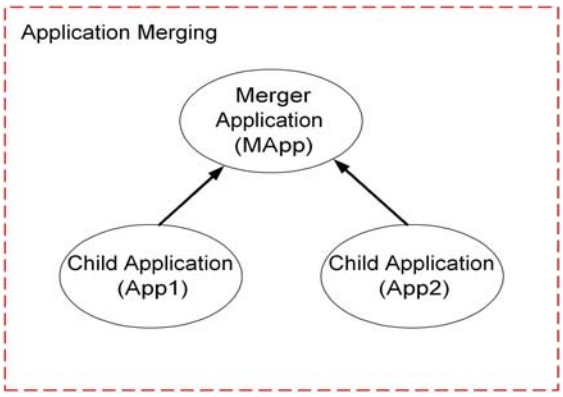
the merged applications are aligned to facilitate a combined functionality in the new merger application. Yet, the virtual merger application can be treated just like any other eCOIN application, and extended with new set of sources, ontology elements, context instances, mappings, and can be merged with other applications. Next we illustrate how the creation of a virtual merger application is accomplished by using the airfare and car rental applications.

3.2 MERGING PROCEDURE & REPRESENTATION IN eCOIN

The virtual merging process in eCOIN is driven by the need to reconcile context models and the conversion functions of the individual applications. In addition, the new context model may be enhanced beyond the union of the individual models because shared understandings in individual domains may clash when combined. We also want to utilize the existing conversion function networks of individual applications, thus these networks need to be bridged to achieve full reuse of the existing mappings.

The merging process in eCOIN roughly corresponds to the flowchart shown in Figure 11, which refers to the following declarations:

Definition (Merger Declarations)



Let A be the application that merges applications A_1 to A_n[6].

- A *merging relationship* that specifies the merger and the merged applications: **merges(A, [A_1,A_2])**

  This is read as: Application A is the *merger* root of applications A_1 and A_2.

- An *isoSemanticType relationship* that specifies the semantic type mappings between the merger and the merged applications: **isoSemanticType(A,A_i,τ,τ_i)**

  This is read as: Semantic type τ in application A and semantic type τ_{ij} in application A_i has compatible modifiers. Note that this declaration does not mean that τ and τ_i are synonyms.

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

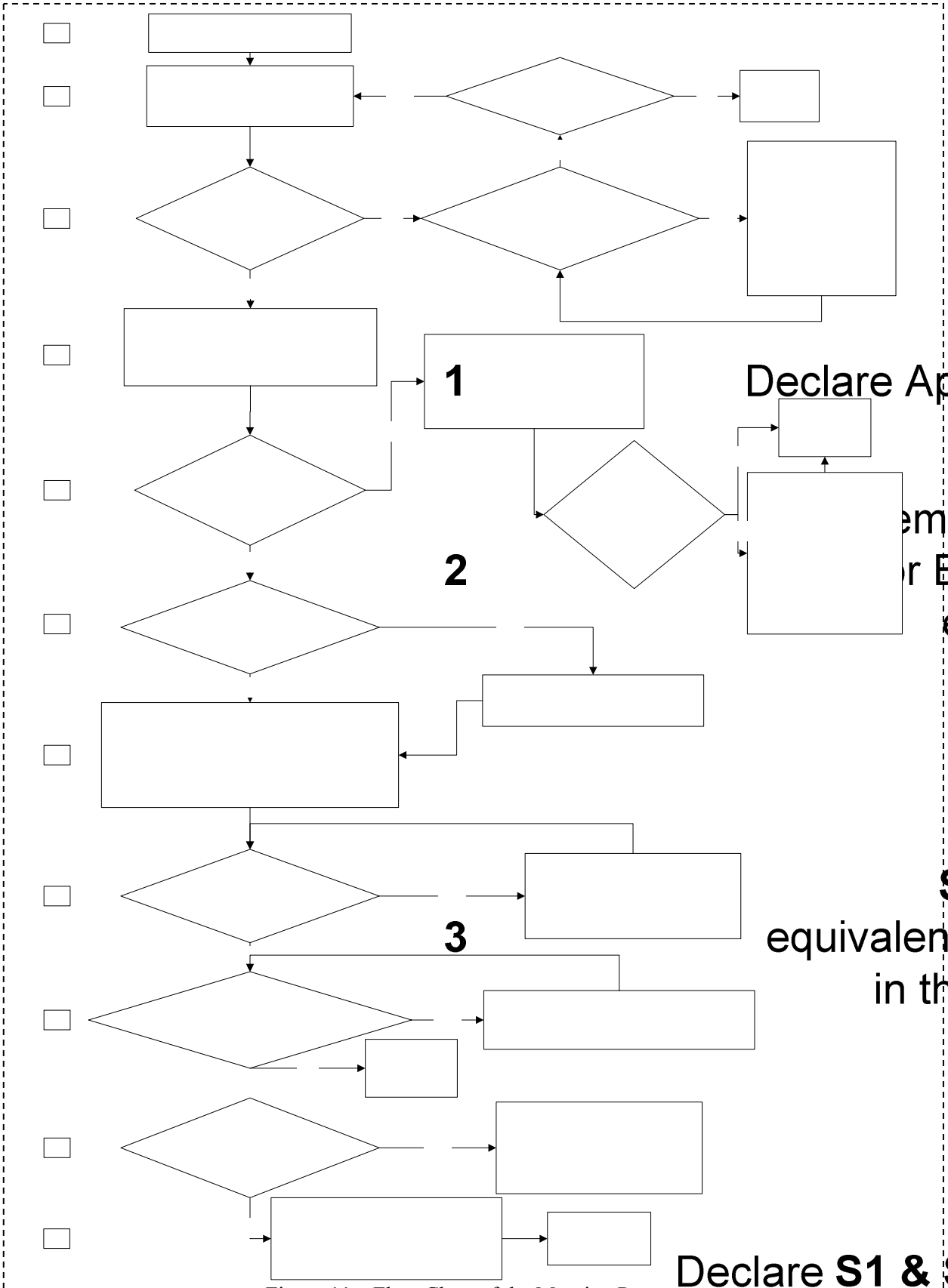


Figure 11 – Flow Chart of the Merging Process

It means that these two concepts have the same set of modifiers. Related but different concepts (e.g. revenues vs. profits), and more specialized or general versions of the same concept (e.g. financials vs. profits) can all qualify to have the same set of modifiers.

- An *isoModifier relationship* that specifies the modifier name mappings between the merger and the merged applications: **isoModifier(A, A_i, m, m_i)**

  This is read as: Modifier m in application A and modifier m_i in application A_i are equivalent modifiers.

- An *isoContext relationship* that specifies the context identifier mappings between the merger and the merged applications: **isoContext(A,A_i,c,c_i)**

  This is read as: Context label c in application A and context label c_i in application A_i are equivalent.

  An *isoAttribute relationship* that specifies the attribute name mappings between the merger and the merged applications: **isoAttribute(A,A_i,a, a_i)**

  This is read as: Attribute a in application A and attribute a_i in application A_i are equivalent attributes.

The above mappings are always specified between the merger and the merged applications, never between merged applications directly. In Figure 12, the result of the merging for the car rental and airfare applications is shown. Below, we discuss some of the important points of this merging process.

UPWARD INHERITANCE

By default all of the elements of the applications to be merged are included in the merger application. When there are equivalent elements, the merger declarations designate, which one of those elements is chosen to be upward inherited. In Figure 12, elements that are upward inherited are shown with bold borders. For the car rental and airfare applications these are accomplished with the following declarations.

Merger Axioms

```
merges(travel, [airFare, carRental]).
isoSemanticType(travel, carRental, date, date).
isoSemanticType(travel, airFare, tax, tax).
isoSemanticType(travel, carRental, airport, airport).
```

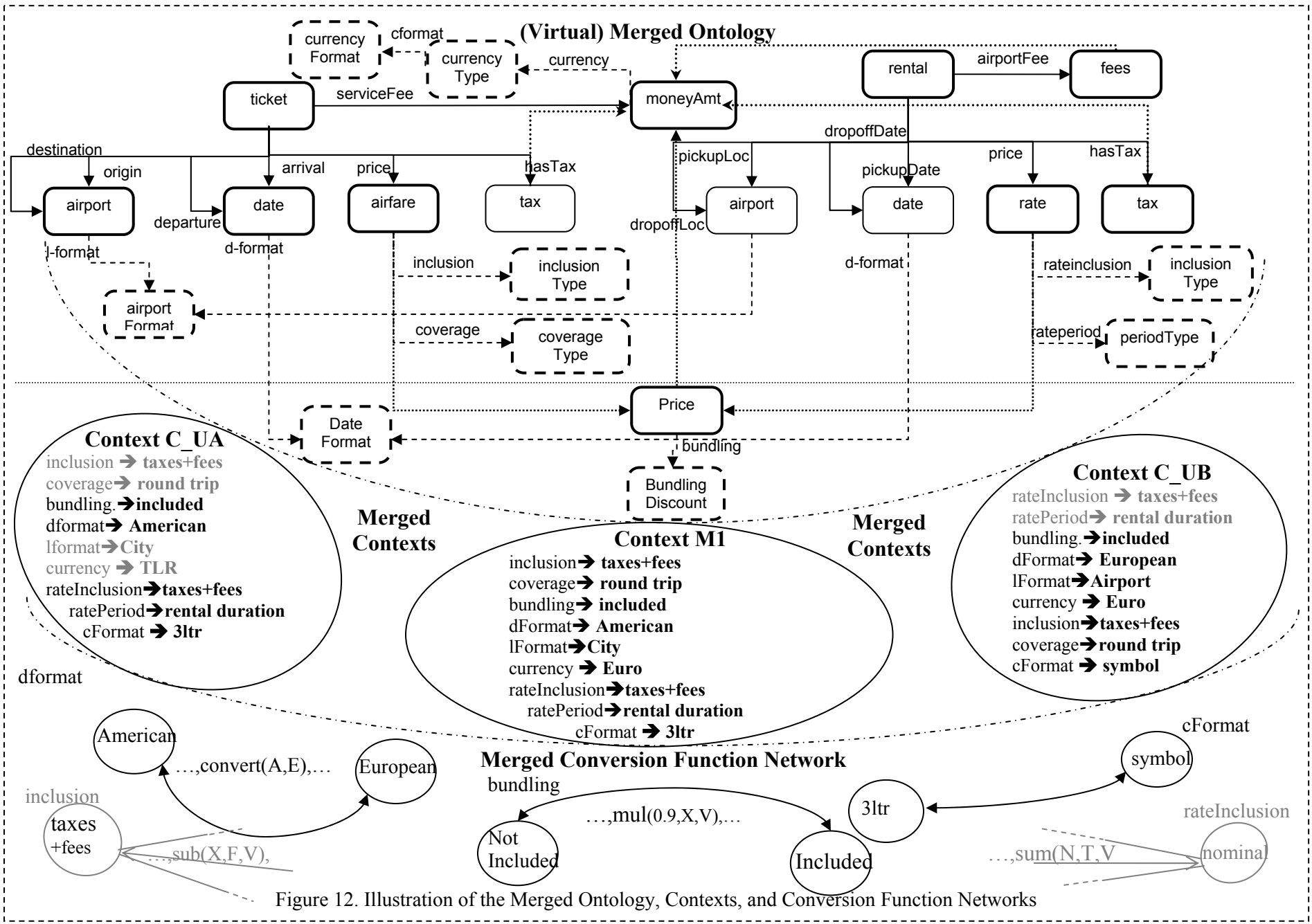


Figure 12. Illustration of the Merged Ontology, Contexts, and Conversion Function Networks

ONTOLOGY EXTENSIONS

Because the merger application is just like any eCOIN application, it can be extended with additional elements to facilitate the merging process or simply enhance the ontology. As shown in Figure 12, the virtual merger ontology has a new semantic type *price*, which acts as a super type of airfare and rate semantic types, and a sub type of the moneyAmt semantic type. In addition to the modifiers it inherits from the moneyAmt, price also has a new modifier of its own called *bundling*. This new modifier is used to represent the previously inapplicable assumption of including the bundling discount or not when rental and airfare is purchased together.

Furthermore, the ontology is extended by introducing a modifier for the date type, because the interpretation of the date type is no longer a shared one after the merger: airfare uses American style dates, whereas the car rental uses European style.

Another interesting extension is seen in the case of defining a modifier of the currency modifier. In this case, we assumed for illustration purposes that the sources and receivers in the car rental application would like to express currencies using the currency symbols (e.g. $, £, etc.). This necessitates the declaration of the cFormat modifier for the currency type to allow variations in the way currencies are expressed. All of these extensions are expressed with the following logical predicates:

```
semanticType(price).
isa(airfare,price).isa(rate,price).isa(price,moneyAmt). isa(fees,moneyAmt).
bundling(price, Context, bundlingType).
dformat(date, Context, dateFormatType).
cFormat(currencyType, Context, currencyFormat).
```

CROSS FERTILIZATION OF CONTEXTS

During merging it is highly likely that new modifiers are introduced for the individual applications. Because the individual applications were previously unaware of these modifiers, their values need to be introduced in the merger application. For example, the values for the data modifier dformat need to be introduced for all existing contexts, because date format had a shared representation and meaning before

the merging. The number of context declarations, however, can be reduced by defining "umbrella" contexts as the supertypes of the application contexts. Below, this is exemplified by first defining c_usa as the umbrella context for the airfare application, and c_europe for the car rental. The context labels of each application are then organized in a sub type hierarchy with the umbrella contexts. Subsequently the modifer values for the umbrella contexts are declared.

Context Hierarchy:

```
isa(c_ct,c_usa).isa(c_ua,c_usa).isa(c_et,c_usa).
isa(c_ub,c_europe). isa(c_cr,c_europe).
```

Context Values:

```
dformat(Date',c_usa,Format')  ←
  Format' = object(dateFormatType,"American",c_usa,constant("American")).
dformat(Date',c_europe,Format')     ←
  Format' = object(dateFormatType,"European",c_europe,constant("European")).
```

Similarly, the rest of the missing modifier values can be declared for the cFormat, bundling lformat, inclusion, coverage, rateInclusion, and the ratePeriod modifiers.

LINKING THE CONVERSION FUNCTION NETWORKS

With the introduction of new modifier values, the conversion function networks of the individual applications need to be aligned, and if need be, extended. For example, the emergence of dformat, bundling, and cformat modifiers necessitates the declaration of three new conversion functions. Some conversion functions, however, can be used without any modification. For example, the carRental application can readily use the currency conversion function of the airfare application without any modifications. The differences in the way currencies are represented in both applications are taken care of by the conversion function defined for the cformat modifier, thus do not create any problems.

In some other cases, the conversion function may not be parameterized like the currency conversion function, thus new nodes may need to be introduced. For example, if a new receiver wants to express the dropoff and pickup locations of cars with zipcodes, the conversion function network for the lformat

modifier can be extended with a new node called “zipcode” and appropriate conversion functions connecting this new node to the network (e.g. to the “city” node).

Furthermore, if the conversion functions use any constructs peculiar to the individual applications, these constructs need to be defined for the objects of the other application. If this cannot be done, the conversion function has to be redefined in the worst case.

3.3 MEDIATION OF THE QUERY OVER THE MERGER APPLICATION

The query Q3, shown in Figure 9, can now be mediated by the eCOIN engine into the following mediated query MQ3:

```
MQ3: SELECT “cheaptickets” as fareprovider, “Lufthansa”, “National”, ((2 * (t.Price + Tax)+5) * eRate +
r.Price  * 34.65) * 0.9 as total
FROM     cheaptickets t, currencyrates, cheaprentals r,
   (select Airport from cityairport where city= “Boston”) cityairport1,
   (select Airport from cityairport where city= “Istanbul”) cityairport2
WHERE DepDate = “06/01/05” and ArrDate=”07/01/05” and   DepCity= cityairport1.Airport and
   ArrCity= cityairport2.Airport  and fromCur= “USD” and toCur= “EUR”  and Date= “05/10/05” and
   Airline=”Lufthansa” and Company=”National” and Class= “Economy” and
   PickDate = “02/06/05” and DropDate=  “01/07/05”  and
   Pickup= cityairport2.Airport and DropOff= cityairport2.Airport
UNION
SELECT “cheaptickets” as fareprovider, Airline, Company, ((2 * (t.Price + Tax )+5) * eRate + r.Price  *
34.65)  as total
FROM cheaptickets t, currencyrates, cheaprentals r,
   (select Airport from cityairport where city= “Boston”) cityairport1,
   (select Airport from cityairport where city= “Istanbul”) cityairport2
WHERE DepDate = “06/01/05” and ArrDate=”07/01/05” and   DepCity= cityairport1.Airport and
   ArrCity= cityairport2.Airport and fromCur= “USD” and toCur= “EUR” and Date= “05/10/05” and
   (Airline<>”Lufthansa” or Company<>”National”) and Class= “Economy” and
   PickDate = “02/06/05” and DropDate=  “01/07/05”  and
   Pickup= cityairport2.Airport and DropOff=cityairport2.Airport
UNION
SELECT “eurotickets” as fareprovider , Airline, Company, (t.Price * eRate + r.Price  * 34.65)  as total
FROM eurotickets t, currencyrates, cheaprentals r,
   (select Airport from cityairport where city= “Boston”) cityairport1,
   (select Airport from cityairport where city= “Istanbul”) cityairport2
WHERE DepDate = “06/01/05” and ArrDate=”07/01/05” and   DepCity= cityairport1.Airport and
   ArrCity= cityairport2.Airport and fromCur= “USD” and toCur= “EUR” and Date= “05/10/05” and
   Class= “Economy” and PickDate = “02/06/05” and DropDate=  “01/07/05”  and
   Pickup= cityairport2.Airport and DropOff=cityairport2.Airport
```

This mediated query reconciles all the conflicts between the sources and the user context c_um. The prices are adjusted to include all taxes and fees; discounted when applicable (note the multiplication by

0.9 in the first subquery to take care of the bundling situation). Car rental prices are adjusted to cover the rental duration; and airfare reflects round-trip prices. Furthermore, all the prices are reported in Euros. When this mediated query is executed, it produces the following non-obvious results, shown in Table 5, which reveal a three way tie between 'British Airways & Hertz', 'Lufthansa & National' and "United Airlines and Hertz".

fareprovider	**Airline**	**Company**	**total**
cheaptickets	Lufthansa	National	1747
cheaptickets	British Airways	Hertz	1747
eurotickets	United Airlines	Hertz	1747
cheaptickets	Lufthansa	Hertz	1775
eurotickets	Delta	Hertz	1791
cheaptickets	British Airways	National	1914
eurotickets	United Airlines	National	1914
eurotickets	Delta	National	1958

Table 5. Results of MQ3

4. RELATED WORK

This paper describes a novel approach to achieving semantic interoperability between ontology based applications, thus relates to the broad ontology integration techniques literature. The origins of ontology integration can be traced back to schema integration, which has been studied extensively since 1980's [Batini et al 86]. Schema integration research produced some guidelines to be used in integrating disparate schemas and semi-automatic tools, but the process could not be fully automated because schema semantics could not be made explicit without human intervention.

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

variety of languages. In our study, we assumed that ontologies were represented using the same ontology language thus avoided the syntactical issues.

Semantic differences such as the ones outlined by [Reed and Lenat 02] and shown in Figure 13, however, offer more difficult challenges as they require human intervention to understand and reconcile the meaning of ontological terms and relationships.

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

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

Figure 13 List of differences between ontologies (Adopted from [Reed and Lenat 02])

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

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

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

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

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

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

The state of the art in ontology merging today is dominated by semi-automatic tools that can analyze ontologies, and guide the user during merging by making suggestions. Examples of well known such tools are Prompt [Noy and Musen 00], Ontomorph [MacGregor et al. 99] and Chiemera [McGuinness et al. 00]. Recently, there are also approaches extending these techniques to Semantic Web [Doan et al. 04].

In these types of semi-automatic matching approaches, which is surveyed in [Rahm and Bernstein 2001], the first step is the *syntactic match* phase in which ontological terms referring to similar objects are identified based on a linguistic similarity measurement. In the simplest case synonyms from a thesaurus can be used. In more sophisticated approaches, a lexical reference system like Wordnet [Miller 95] can be used to identify similar terms through the use of richer semantics that involves relationships linking different synonyms sets.

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

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

understanding each ontology before doing the merging, and fully automatic approaches remain as a dream.

5. LIMITATIONS AND FUTURE WORK

Our eCOIN approach described in this paper dramatically reduces the amount of work needed in creating ontologies by shifting the focus from agreeing on the exact meaning of the terms to identifying possible dimensions of conflicts. There is still, however, some amount of coordination needed in determining what these dimensions will be and what values can be used for those dimensions. If eCOIN is to be used in a completely open environment like the Semantic Web, there needs to be further research in determining how some of the underlying restrictions of coordination can be overcome or reduced.

Furthermore, in the current conceptualizations of the Semantic Web, eCOIN constructs such as contexts, and conversion function networks are not explicitly accounted for. In order to translate the conceptual framework of eCOIN to that of the Semantic Web, further work is needed to find out how to represent contexts, conversion function networks using the available Semantic Web constructs such as ontologies, and rules [Grosof 2001]. Preliminary results of such an effort can be found in [Tan et al. 2005].

There are also limitations in the ontology merging tools we developed so far, and further work is needed. In eCOIN, we use a semi-automatic tool called CLAMP [Kaleem 03, Anwar 04] (a screen shot is shown in Figure 14), to guide the user during the application merging process. This tool, however, does not currently use any of the match techniques described in the literature, but facilitates the merging process by generating the underlying logical predicates through visual interfaces. Incorporating the existing match techniques into this tool would further help the users in dealing with large scale applications.

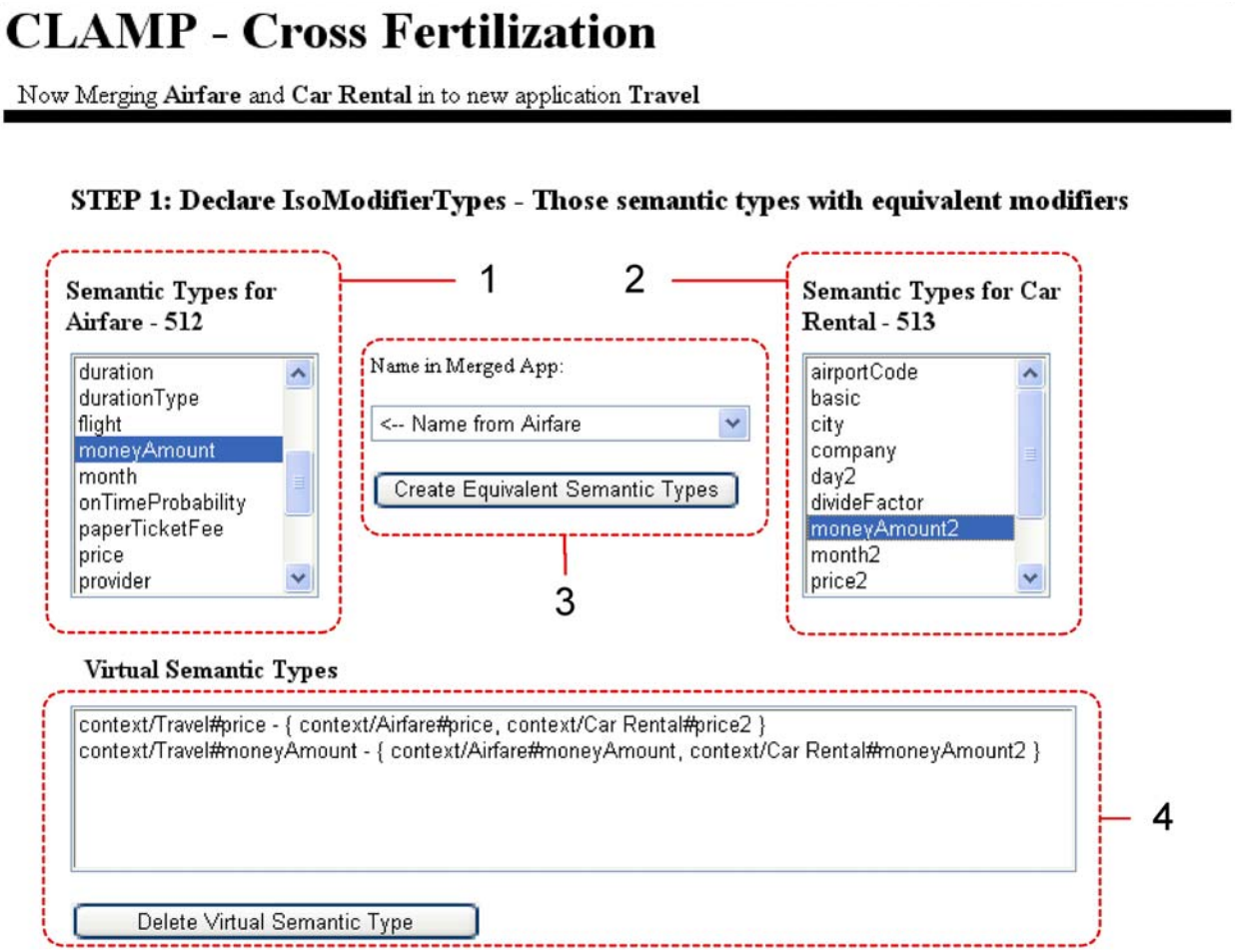


Figure 14. CLAMP Tool for Application Merging in eCOIN

6. CONCLUSION

We believe that achieving semantic interoperability between diverse information sources and receivers should have the dual purpose of: (1) *reconciling* semantic heterogeneity across information sources; and (2) *supporting* semantic heterogeneity across information receivers. In this paper, we offer a novel approach that satisfies this dual purpose by representing the semantic heterogeneity explicitly based on a context model. This context model, associated with an ontology, allows a mediation engine to detect and reconcile semantic conflicts between the sources and receivers on the fly and present the data to the users in the way they want.

Our approach has also a number of system level benefits. First, ontology developers do not have to standardize the exact meaning and representation of ontological terms; but only need to agree on generic

identities without exposing the specific details. A major advantage of this approach is that ontology developers frequently find it straightforward (if not necessarily "easy") to agree on the generic concepts; it is getting all the precise details worked out that creates a lot of the work. Moreover, it's often the case that differences in these precise details are only discovered later (sometimes even after the system is in operation). The eCOIN approach enables these details to be factored out, reducing the amount of work needed to introduce these details all at once.

Second, because the ontology is impartial to the precise semantics defined in the various contexts, mappings are not defined between the sources and the ontology as it is done in most current approaches to information integration. Instead, mappings are structured with respect to a context model and defined for each modification dimension as a *conversion function network*. This modularization of mappings allows a mediator to create custom point-to-point translations between contexts by selecting or composing appropriate mappings from the conversion function network.

The utility of our approach, however, would be much more limited if we did not have a methodology to achieve interoperability between applications based on the eCOIN framework. With this study, we described a distinct approach to enabling large scale interoperability through the virtual merging of multiple eCOIN applications. The virtually merged application creates the illusion of a single system that can access sources across domains; accomplishes cross fertilization of contexts and conversion functions; and offers value added benefits beyond what the underlying applications can provide.

We have implemented these ideas in a prototype implementation [Firat 2003] using the Eclipse Prolog engine [Cheadle et al. 2003] and procedural programming languages. This prototype provides mediated access to traditional databases, as well as semi-structured web sites, and web services; creates and maintains metadata that are used in eCOIN through graphical interfaces, and supports merging multiple applications. With a working prototype implementation and sound theoretical basis, we claim that eCOIN provides an elegant and effective solution for reconciliation of semantic conflicts.

6. REFERENCES

[ANWAR, F. 2004] "Virtual Merging of Ecoin Applications and Guidelines for Building Ontologies", *Masters of Engineering Thesis*, Massachusetts Institute of Technology, June, 2004.

[BATINI, C., LENZERINI, M., NAVATHE, S. B. 1986] "A Comparative Analysis of Methodologies for Database Schema Integration", *ACM Computing Surveys*, vol. 18(4), p. 323-364.

[BERNERS-LEE, T., HENDLER, J., LASSILA, O. 2001] "The Semantic Web", *Scientific American*, May 2001.

[CHALUPSKY, H. 2000] "OntoMorph: a translation system for symbolic knowledge", In *Proceedings of Seventh International Conference on Knowledge Representation and Reasoning*, p. 471--482, San Francisco, California, Morgan Kaufmann

[CHEADLE, A. M., HARVEY, W., SADLER, A.J., SCHIMPF, J., SHEN, K., AND WALLACE M. G., 2003]. "ECLiPSe: An Introduction", *IC-Parc Imperial College London, Technical Report*.

[DOAN A, MADHAVAN J, DHAMANKAR R, DOMINGOS P, HALEVY A. 2004] "Learning to match ontologies on the SemanticWeb", *The VLDB Journal*, 12 p. 303--319, 2004.

[FIRAT, A. 2003] "Information Integration Using Contextual Knowledge and Ontology Merging", *Ph.D. Thesis*, Massachusetts Institute of Technology.

[FIRAT, A., MADNICK, S., AND SIEGEL, M. 2000] "The Caméléon Web Wrapper Engine", In *Proceedings of the VLDB2000 Workshop on Technologies for E-Services*, p. 1-9.

[FRIDMAN NOY, N., MUSEN, M. A. 2000] "PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment", *AAAI/IAAI*, 2000, p. 450-455.

[GOH, C. H. 1997] "Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Systems", *Ph.D. Thesis*, Massachusetts Institute of Technology.

[GROSOF, B. 2001]. "Representing E-Business Rules for the Semantic Web: Situated Courteous Logic Programs in RuleML", In *Proceedings of the Workshop on Information Technologies and Systems* ,WITS '01.

[GRUBER, T. R. 1993] "A Translation Approach to Portable Ontology Specifications", *Knowledge Acquisition*, vol. 5, p. 199–220.

[HALEVY, A. 2000]. "Theory of Answering Queries Using Views", *ACM SIGMOD* Record, vol. 29(4), p. 40-47.

[HALEVY, A.Y., MADHAVAN, J., AND BERNSTEIN, P. A. 2003.] "Discovering Structure in a Corpus of Schemas", *Data Engineering Bulletin*, September 2003, p. 26-33.

[IVES, Z., HALEVY, A., MORK, P., AND TATARINOV, I. 2004] "Piazza: Mediation and Integration Infrastructure for Semantic Web Data", *Journal of Web Semantics*, Vol. 1 No. 2, February 2004, p. 155-175.

[KALEEM, M. 2003] "CLAMP: Application Merging in the Ecoin Context Mediation System Using the Context Linking Approach", *Masters of Engineering Thesis*, Massachusetts Institute of Technology, August, 2003.

[LENZERINI, M 2002] "Data Integration: A Theoretical Perspective", *PODS* 2002: 233-246

[MCGUINNESS, D.L., FIKES, R., RICE, J. AND WILDER, S. 2000] "An Environment for Merging and Testing Large Ontologies", In: *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning* (KR2000), Breckenridge, Colorado.

[MILLER, G. 1995]. "WordNet: a lexical database for English", *Communications of the ACM*, vol 38(11), p. 39–41.

[NILES, I & PEASE A. 2001] "Towards A Standard Upper Ontology", In *Proceedings of FOIS 2001*, Ogunquit, Maine, USA.

[POTTINGER R. AND BERNSTEIN P. A. 2003] "Merging models based on given correspondences", In *VLDB Conference*, 2003.

[RAHM, E., BERNSTEIN, P. A. 2001] "A survey of approaches to automatic schema matching", *VLDB Journal*, vol. 10(4), p. 334-350, 2001.

[REED, S. AND LENAT, D. 2002] "Mapping Ontologies into Cyc", *AAAI-2002 Workshop on Ontologies and the Semantic Web*, http://reliant.teknowledge.com/AAAI-2002/Reed.pdf

[SCIORE, E., SIEGEL, M., AND ROSENTHAL, A. 1994] "Using Semantic Values to Facilitate Interoperability Among Heterogeneous Information Systems," *ACM Transactions on Database Systems (TODS)*, Volume 19 Issue 2, June 1994.

[SIEGEL, M., MADNICK, S. 1991] "A Metadata Approach to Resolving Semantic Conflicts, In *Proceedings of the Seventeenth International Conference on Very Large Databases*, pp. 133-145.

[TAN P., MADNICK, S., TAN K. 2005] "Context Mediation in the Semantic Web: Handling OWL Ontology and Data Disparity Through Context Interchange", *Lecture Notes in Computer Science*, Volume 3372, p. 140.

[WAND, Y., STOREY, V., AND WEBER, R. 1999] "An ontological analysis of the relationship construct in conceptual modeling", *ACM Transactions on Database Systems* 24, 494 -528.