

Semantic Interoperability through Context Interchange: Representing and Reasoning about Data Conflicts in Heterogeneous and Autonomous Systems*

Cheng Hian Goh[†] Stuart E. Madnick Michael D. Siegel
Sloan School of Management
Massachusetts Institute of Technology
Email: {chgoh,smadnick,msiegel}@mit.edu

Abstract

This paper describes a strategy for the construction of large-scale heterogeneous systems which combines the benefits of both tight-coupling systems (insulating users from the complexity of semantic heterogeneity thus promoting greater usability) and loose-coupling systems (allowing changes in individual components to be contained locally thus assuring the long-term viability of the system as a whole). We define a new data model (COIN) based on a deductive and object-oriented language with which knowledge of data semantics is represented in the form of *enriched schemas*, *contexts* and underlying *domain models*. Query mediation in this framework is driven by a top-down evaluation strategy which yields a query plan describing what sources are used and what conversions must be applied in mediating semantic conflicts. The COIN data model induces a novel dichotomy between schemas and contexts thus permitting knowledge of data semantics to be shared and reused across different systems in similar environments. This in turn provides various options for querying disparate sources, with each offering a different level of transparency depending on the needs of the users. The richness of the model offers receivers more flexibility, both in defining what disparities should count as conflicts, and how conflicts should be resolved.

Keywords: heterogeneous databases, information infrastructure, knowledge and data modeling, mediators, semantic interoperability.

*This work is supported in part by ARPA and USAF/Rome Laboratory under contract F30602-93-C-0160, the International Financial Services Research Center (IFSRC), and the PROductivity From Information Technology (PROFIT) project at MIT. Further information, including electronic versions of papers originating from the Context Interchange Project, is available on the WWW at <http://rombutan.mit.edu/context.html>.

[†]Financial support from the National University of Singapore is gratefully acknowledged.

1 Introduction

Almost every statement we make is imprecise and hence is meaningful only if understood with reference to an underlying context which embodies a number of hidden assumptions. This anomaly is amplified in databases due to the gross simplifications we make in creating a database schema. For example, a database may record the fact

```
salary('Jones', 2000)
```

without ever explaining what “2000” means (what currency and scale-factor is used?), what is the periodicity (is this the daily, weekly, or monthly wage?), or what constitutes the person’s salary (does it include year-end bonuses? what about overtime pay?). The real problem occurs when sources and receivers¹ maintain different assumptions about the data which are being exchanged: a receiver formulates a query and interprets the answers returned in a certain context, whereas the query is executed by source(s) which most likely provides the answers in a completely different context. Under these circumstances, *physical connectivity* (the ability to exchange bits and bytes) does not necessarily lead to *logical connectivity* (the ability to exchange meaningful information). This situation deteriorates rapidly when there is a multiplicity of both sources and receivers.

The problem which we have just described is traditionally referred to as that of achieving *semantic interoperability* among “heterogeneous databases” [18,20], or sometimes, “multidatabase systems” [2,3,10]². In most instances, solutions proposed in the current literature can be classified into one of two generic strategies: reconcile conflicting data representations by translating them to some canonical form in one or more shared schema(s) against which queries are issued [1,12]; or, provide users with a powerful data manipulation languages (e.g., MDSL [13]) with which data translations can be specified as part of a query. Systems constructed using one of the two strategies are said to be *tightly-coupled* or *loosely-coupled* respectively.

This paper describes a third strategy, called *Context Interchange*, aimed at supporting the construction of large-scale heterogeneous systems³ by combining the benefits of both tight- and loose-coupling systems: the former allowing users to be insulated from semantic conflicts between data in disparate systems, while the latter allowing individual components to evolve with little

¹Sources refer to databases, data feeds, and other applications which provide structured data upon request; receivers refer to users, consolidating databases (e.g., data warehouses), and applications that make these requests.

²The careful reader will however notice that our problem statement accord greater symmetry to sources and receivers whereas the database literature tends to focus exclusively on conflicts between disparate data *sources*: just as sources are autonomous and heterogeneous, receivers (applications and users) differ widely in their assumptions of how data should be interpreted and often are equally resilient to changing their expectations. Hence, semantic conflicts may still occur even if there is merely one source and one receiver.

³which makes it substantively different from earlier reports [19,21] where the focus is on data exchange in a single-source, single-receiver system.

impact on the larger system. At the same time, this strategy also encourages the incremental buildup of an information infrastructure, in the form of knowledge of the underlying domain and of data semantics in different environments, with which addition of new system components can be made progressively easier.

A key contribution of this paper is a collection of technical innovations which make this integration strategy viable. A new data model, called COIN, is introduced as the underlying formalism for the representation of domain knowledge and data semantics. This induces a dichotomy between schemas and contexts: a distinction which proves to be invaluable in promoting the sharing and reuse of knowledge concerning the meaning of data in different environments. The richness of the model also allows receivers to define what disparities should count as conflicts, and how these can be resolved. Finally, query mediation within this framework is accomplished using a top-down evaluation strategy resulting in a query plan which describes what sources and conversion operations are needed to mediate data exchanges at each step.

The rest of the paper is organized as follows. The next section provides the motivation for the Context Interchange approach by comparing the strengths and weaknesses of existing loose- and tight-coupling integration strategies. This is followed by a brief presentation of the Context Interchange architecture which provides an overview of the various system components and their relationships to one another.

Section 3 presents the COIN data model which is based on a deductive and object-oriented formalism. Major differences between COIN and the *semantic value model* [19] are: the parsimonious integration of conversion knowledge in the *domain model*, the distinction between *schemas* and *contexts*, and the relationship of the latter two to the shared *domain model*.

Section 4 describes the various options a user has in formulating a query in a Context Interchange system, each providing a different level of transparency. We present an example which illustrates how a query plan is generated by the context mediator. This strategy represents a significant departure from the traditional single-source or multidatabase query processing approaches: it takes into account conversion operations needed for mediating semantic conflicts; and that these operations may necessitate access to additional sources which could embody further conflicts.

Section 5 highlights a number of extensions to the model. We introduce the notion of *context hierarchies* whereby context assertions can be represented at different levels of abstraction such that constraints in some “higher-level” context can be inherited and even overridden by “lower-level” contexts. This overriding extends not only to constraints on semantic values but also function definitions hence providing the mechanism for different receivers to use customized conversion functions in distinct contexts. The same strategy allows users to specify whether

DATABASE disclosure_db, RELATION disc

Company	Profit	CountryIncorp
Daimler Benz Corp	10,000,000	Germany
⋮	⋮	⋮

DATABASE worldscope_db, RELATION ws

Name	Net_Income	Country
Daimler-Benz AG	6,700	Germany
⋮	⋮	⋮

Figure 1: Snapshots of databases for the supporting example.

a differing feature is, in fact, a conflict by selectively specifying what are the “modifiers” of a semantic value.

The last section summarizes the contribution of this paper and highlights a number of open issues which are being researched.

2 Context Interchange versus Traditional Integration Strategies

Consider the dilemma faced by a user retrieving data from two disparate (but real!) databases, `worldscope_db` and `disclosure_db` shown in Figure 1. Suppose the user is interested in “German companies with a net income < 8000”. At first glance, it is not obvious how the data should be interpreted: does “Daimler-Benz AG” and “Daimler Benz Corp” refer to the same entity? If so, which database is reporting the truth concerning its profitability? It turns out that in this instance, both company names are variants of the same entity. Moreover, `disclosure_db` reports a company’s profit using the currency for the country of incorporation (in this case, Deutsch Marks for Germany) and uses a scale-factor of 1, whereas `worldscope_db` furnishes all information in U.S. Dollars using a scale-factor of 1000. In posing a query, the user is not “context-free” either: in this example, the user has made implicit assumptions on what “8000” means, whether or not he is conscious of it: moreover, if the query were to be executed directly on both databases without any mediation, “Daimler Benz Corp” in `disclosure_db` would not qualify as an answer⁴.

In a *loose-coupling* system, a user will need to figure out that the above-mentioned conflicts exist and write a query augmented with the necessary conversion operations. For example, appropriate currency and scale-factor conversions will have to be applied to results obtained from `disclosure_db`. This process may require additional information which is unavailable from

⁴While the conflicts presented in the example may seem trivial and well-understood, they serve to highlight several of the weaknesses in classical integration approaches. The kinds of conflict we dealt with in this example are deliberately confined to simple cases for pedagogical reasons; this should not be construed as a limitation on the kinds of problems addressable by the Context Interchange strategy. Examples involving more sophisticated conflicts will be introduced in subsequent sections.

the original source (e.g., what is the current exchange rate for Deutsch Marks and U.S. Dollars) and the user will have to figure out where this information can be found. Largely as a response to these problems, advocates of the *tight-coupling* approach suggest these conflicts should be resolved a priori by defining a view which presents a single canonical representation. We may, for instance, define a view in which all monetary information is reported using U.S. Dollars by encapsulating the currency conversion function in the view mapping⁵.

We have three main objections to the tight-coupling strategy. First, the resulting view or *shared schema* is tightly-coupled to the underlying component systems suggesting that the shared schema needs to be updated each time the system evolve (as when new sources are added) or when data semantics of existing systems change. Second, by making data conversion and aggregation part of the view definition, the semantics of data in underlying systems is effectively encapsulated *procedurally*; the consequence is that we will no longer be able to pose “meta-level queries” of the form: “what currency is this data reported in?”. Procedural encapsulation of semantics also does not lend well to query optimization: if a source reports data in Deutsch Marks and the user in fact wants the information in British Pounds, it makes little sense to first convert the data to U.S. Dollars. A user also may prefer a different conversion approach to that dictated in the view definition (e.g., using a historical exchange rate as opposed to the prevalent one); unfortunately, little can be done in this case except for creating a different view definition. Finally, the tight-coupling approach provides little opportunities for sharing and reuse of meta-data. For example, these databases report a vast amount of financial information (e.g., total assets, revenues, expenses, etc) having many common characteristics (e.g., currency and scale-factor). In the view definition for the shared schema, attributes are integrated on a piece-meal basis and the same conversion operations will have to be specified for each attribute; this is not only cumbersome but presents the possibilities of inconsistencies when the view definition needs to be updated (e.g., when semantics of underlying data change). Each source is also integrated independently even though different sources situated in the same environment may share similar assumptions of the semantics of underlying data.

The preceding criticisms of the traditional integration strategies constitutes the primary motivation for the Context Interchange strategy⁶. The question which falls out of the above discussion is thus: “Can we achieve semantic interoperability at the level of transparency as in tightly-coupled systems, and yet retain the flexibility of loosely-coupled systems?” The remainder of this section gives an overview of the components of a Context Interchange system aimed at illustrating the general solution approach, while postponing detailed discussions of the underlying

⁵A similar example exemplifying this strategy is reported in [1] for the Pegasus multidatabase system.

⁶A more detailed presentation of the above arguments has been reported in [7].

innovations to subsequent sections.

The architecture of a Context Interchange system shown in Figure 2 differs from classical loose- and tight-coupling systems in having four additional components: a *domain model*; a collection of *contexts*; a collection of *enriched schemas*; and a specialized query processor called a *context mediator*.

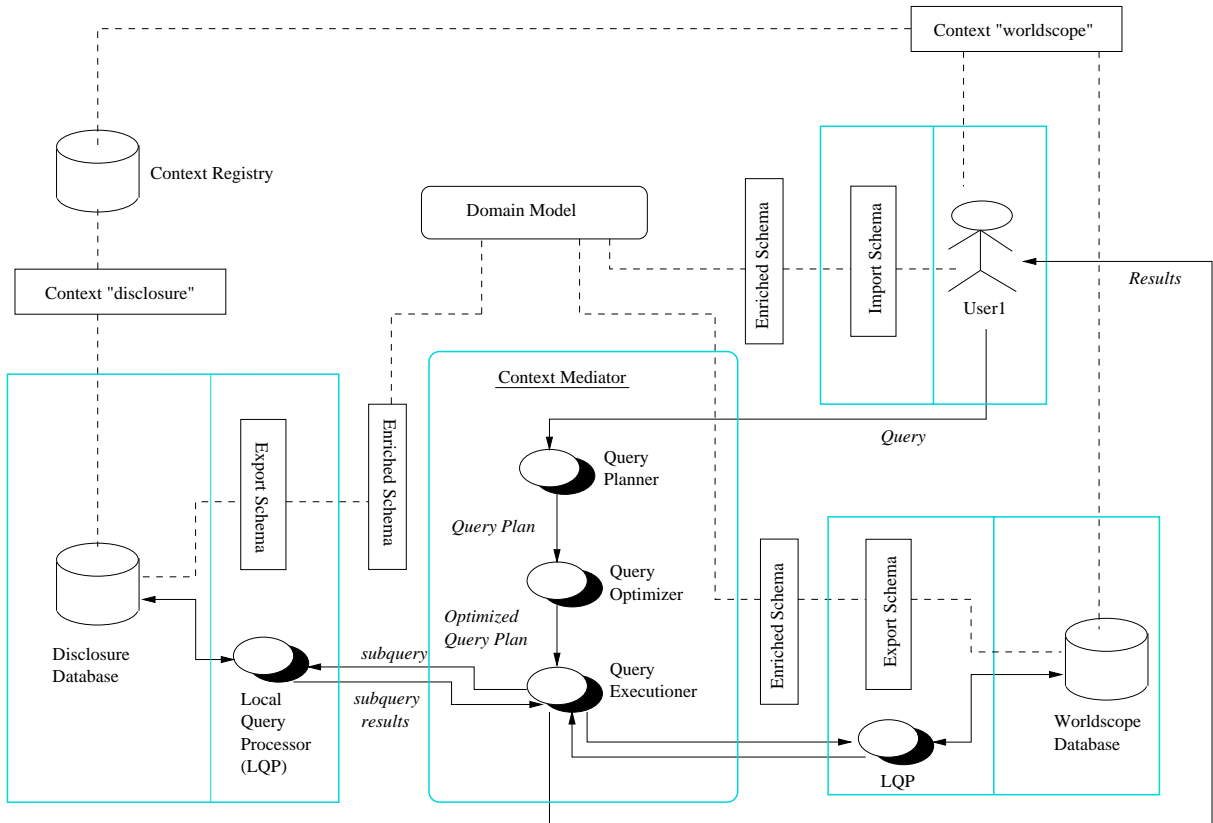


Figure 2: The architecture of a Context Interchange System

The *domain model* defines a conceptual model of the underlying functional domain and provides a basis (i.e., a shared vocabulary) with which the meaning of data in different sources (and receivers) can be described⁷. This “explication” is accomplished in two ways: by having sources and receivers identify with some *context*, and in augmenting the corresponding export and import schema to yield an *enriched schema*.

The *context* associated with a source or receiver introduces additional constraints or as-

⁷In some AI circles, this is sometimes referred to as the collection of *shared ontologies*. Our domain model may in fact be configured, upon requests, from a library of ontologies [8]. At this time however, we simply view the domain model as a conceptual model for a given domain, a concept well-understood in the database circles.

sertions with reference to the domain model. Assertions in a context are independent of the schemas corresponding to the sources and receivers they are associated with. This permits distinct components to share the same context: in our earlier example, the user, who under normal circumstances would operate on data furnished by `worldscope_db` and whose applications are set up to function under those premises, may require that returned answers always comply with the meanings adopted in `worldscope_db`; hence we say that both have the same context (say, `worldscope`). An *enriched schema*, on the other hand, is used in augmenting an export or import schema⁸ by serving as the link between the semantically-impovertish relational structure in database schema and the semantically-rich domain model. Roughly speaking, this amounts to mapping attributes in a schema to objects in the domain model.

The final component of the Context Interchange architecture, the *context mediator*, detects and reconciles semantic conflicts arising from the exchange of data between sources and receivers in disparate contexts. A query originating from a receiver is assumed to be formulated in the receiver's context (unless otherwise stated), and the goal of the context mediator is to ensure that data returned to the user complies with the constraints in the user's context. To facilitate this task, the context mediator draws on the context and schema definitions of all contributing sources and that of the receiver, as well as domain knowledge found in the domain model. The *query planner* figures out what conversions are needed and yields a *query plan* which is fine-tuned by a *query optimizer*. A *query executioner* completes the process by distributing subqueries to the respective sources and collating the partial results, executing the necessary data conversions, and returning the final answer to the receiver.

3 Representing the Meaning of Data

Following a brief introduction to F-logic, we provide a description of the data model underlying the integration strategy. The relationships between the domain model, contexts and schema forms the subject of the next subsection. In order that we do not get overwhelmed with complexity, our discussion here is focused on a small number of simple conflicts. The subsequent sections build on this scheme by extending the discussion to more complex scenarios which will demonstrate more fully the extent of the Context Interchange strategy.

⁸We use *export schema* to refer to a view exported by a data source which defines the underlying data set which it is willing to share with others, and *import schema* to refer to a user view against which queries are formulated. Without any loss of generality, we assume these to be relational schema.

3.1 F-Logic: A Brief Tour

Our formulation of the data model underlying semantic representation is based on a deductive and object-oriented language, called *F-logic* [11], which provides an elegant integration of many important features of logic and object-orientation. We offer a quick overview of the syntax and (informal) semantics of F-logic in this subsection. It is obviously not viable to present all the features of F-logic in this paper and we shall focus only on aspects which are relevant to our discussion.

Objects in F-logic are identified via *logical object ids* (oids): an *id-term* can either be *ground* (i.e., variable-free), or obtained through the application of *constructors* (function symbols) to one or more id-terms. Objects can be arranged in a generalization hierarchy through the use of *isa-expressions*: hence if c, d and o are id-terms, then $c :: d$ denotes that c is a subclass of d , whereas $o : c$ denotes that o is an instance of the class represented by object c . An *object molecule* takes the form $o[a \text{ ;'-separated list of method expressions}]$ where each method expression can be either a *signature expression* or a *data expression*. The *signature expression* $t[m@t_1, t_2 \Rightarrow \{t_3, t_4\}]$ states that the method m , applied to some instance of t taking arguments of type t_1 and t_2 returns a value which is of type t_3 and t_4 . The single-headed arrow indicates that m is a functional (single-valued) method. If m is a set-valued function, we would replace \Rightarrow with $\Rightarrow\Rightarrow$ instead. Like signature expressions, *data expressions* can be single-valued (*scalar*) or set-valued. The scalar data expression $o[m@o_1 \rightarrow v]$ states that the method m , when applied to object o with arguments o_1 returns the value v . Similarly, we may write $o[m@o_1 \rightarrow\rightarrow \{v_1, v_2\}]$ for set-valued data expressions. Finally, *predicate terms* of the form $p(a_1, a_2)$ are also allowed in F-logic to facilitate its integration with relational languages.

Sentences in F-logic, called *F-formula*, are built up recursively using the above primitives. More precisely, an object molecule is a (molecular) F-formula, and more complex F-formulae can be constructed by simpler ones through the use of logical connectives \vee , \wedge , and \neg , as well as the existential (\exists) and universal (\forall) quantifiers. It is sometimes convenient to combine several formulae together even though they add no expressive power. For example, the formula $o1:t[m1 \rightarrow o2[m2 \rightarrow o3]]$ is equivalent to $o1:t \wedge o1[m1 \rightarrow o2] \wedge o2[m2 \rightarrow o3]$.

In this paper, we are interested only in F-formulae which are Horn-like, with possibly negated expressions in their bodies⁹. Following the terminology in the deductive database literature, we will refer to these as *normal Horn formulae*: i.e., these are F-formulae of the form

$$h \leftarrow e_1 \wedge e_2 \wedge \dots \wedge e_m$$

⁹As was demonstrated in [15], this is not a restriction on the expressiveness of the language since it is at least as powerful as FOPC. However, these constraints make them amenable to efficient query processing.

where h is some expression (referred to as the *head*), and each of e_i ($i = 1 \dots m, m \geq 0$) is an expression or the negation of an expression; the entire conjunct to the right of ‘ \leftarrow ’ is sometimes referred to as the *body* of the formula. (Where there is no ambiguity, we sometimes refer to a normal Horn formula as simply a *rule*.) Negation here refers to *negation-by-failure* as defined in [4], which means that the negated expression $\neg t$ is true whenever we are not able to prove t to be true. We shall further assume that rules are *non-recursive*. This implies that top-down evaluation of any query (using a naive backward-chaining procedure) is guaranteed to terminate.

3.2 Domain Modeling using COIN

As noted earlier, enriched schemas, contexts, and the underlying domain model collectively provides the knowledge of data semantics needed by the context mediator. These three components share a common representational formalism: a data model, called COIN (CONTEXT INTERCHANGE), which is inspired by the notion of semantic values introduced in [19], but having a number of important extensions.

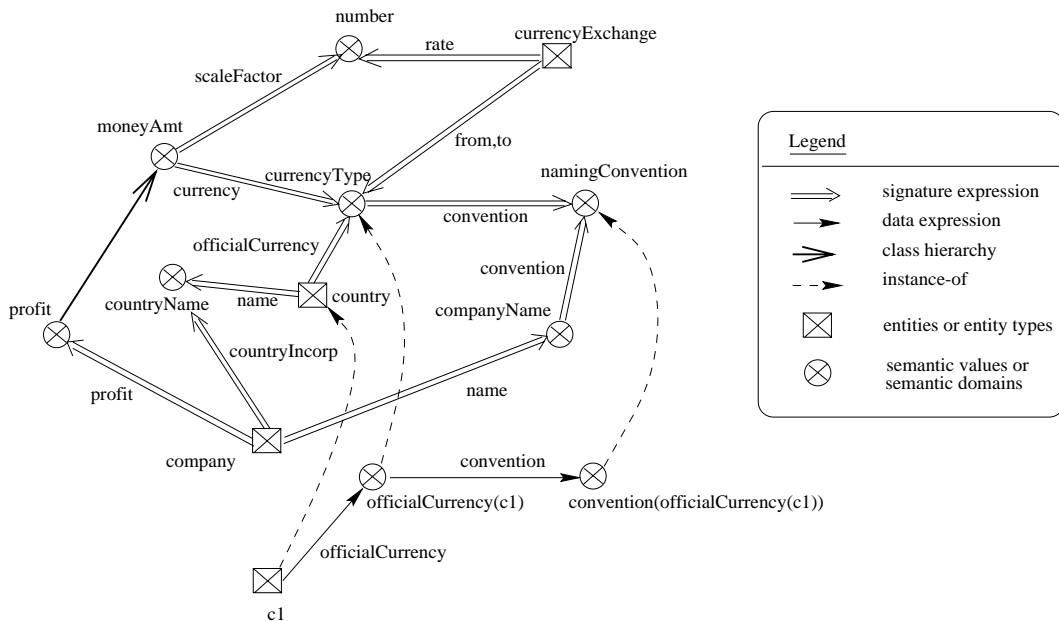


Figure 3: Fragment of a domain model for financial reporting.

Figure 3 shows a fragment of a domain model for an integration scenario in which financial reporting data is being exchanged across disparate systems. Each node in the graph represents an object. In the tradition of semantic data models [9], we distinguish between abstraction objects representing “things” in the real-world and those which are descriptive features or “attributes”

of those things; we refer to these as *entity types* (of which instances are simply *entities*) and *semantic domains* (respectively, *semantic values*). Graphically, entities and entity types are represented using squares whereas semantic values and domains are rendered as circles. As it is in F-logic, we do not emphasize the dichotomy between types (classes) and instances. Directed edges between nodes captures relationships between objects. The four different types of edges in Figure 3 corresponds to the two types of isa-expressions (subclass and instance-of), signature expressions, and data expressions. The same knowledge can be more concisely represented as a collection of normal Horn formulae which is shown in Figure 4. The graphical notation however often provides better intuition to the structure of the domain. For instance, we can readily identify `profit` to be a subclass of `moneyAmt`, or that the entity type `company` has a bunch of attributes including `name` (which instances are drawn from the semantic domain `companyName`), `profit`, and `countryIncorp`.

```

% simple domains
number::semanticDomain.  namingConvention:semanticDomain.
countryName::semanticDomain.
% "non-simple" domains and modifier signatures.
currencyType::semanticDomain[convention=>namingConvention].
moneyAmt::semanticDomain[scaleFactor=>number; currency=>currencyType].
profit::moneyAmt.
% entity types and attribute signatures
currencyExchange::entityType[from=>currencyType; to=>currencyType;
    rate=>number].
country::entityType[name=>countryName; officialCurrency=>currencyType].
company::entityType[name=>companyName; profit=>profit;
    countryIncorp=>countryName].

```

Figure 4: Normal Horn formulae corresponding to the domain model shown in Figure 3.

As mentioned, entities in our model correspond to “real world” objects, having a number of (scalar-valued) *attributes* which range over semantic domains. Our point of departure from classical semantic data models lies in our treatment of *semantic domains (values)*: a semantic value in our model is a complex object which has associated with it a *print-value* (or simply *value* if there is no ambiguity) and zero or more (single-valued) *modifiers*. Modifiers are thus the “attributes” of semantic domains which range over other semantic domains. In our financial reporting example, the semantic domain `moneyAmt` has two modifiers `scaleFactor` and `currency`; the modifier `currency` takes on values drawn from yet another semantic domain `currencyType`, which has one modifier called `convention`. Modifier `scaleFactor` takes on values from the semantic domain `number`; the latter has no modifiers associated with it and is referred to as a

simple domain (the instances of which are *simple values*).

For any scalar (i.e., single-valued) method `m` defined on an object (i.e., entity or semantic value) `o`, the semantic value returned by invoking `m` on `o` can be referred to using the (unique) id-term `m(o)`. Hence, if we define an instance of `country` called `c1`, we may refer to the `officialCurrency` of `c1` to be `officialCurrency(c1)` as shown in Figure 3. Recursively, `officialCurrency(c1)` has a modifier (`convention`) which can be referred to as `convention(officialCurrency(c1))`¹⁰.

To emphasize the role modifiers play in altering the interpretation of a given semantic value, consider an instance `d` of some `date` type which have the value “01/02/96” and the modifier `format`. The interpretation of `d` depends critically on what `format(d)` means. If `format(d)` is a simple value and has print-value “DD/MM/YY”, then `d` refers to “Feb 1, 1996”; if it has print-value “MM/DD/YY”, then we will interpret it to mean “Jan 2, 1996”. That modifiers can be filled with objects (*modifier-values*) which are semantic values themselves is essential since there is not always agreement among heterogeneous sources on what information should be treated as data or meta-data. For example, one data source may report all profit amounts in the local currency (in which case currency is not explicitly represented in the source and constitutes a piece of meta-data), while another might report the currency used in reporting the profitability of each company (in which case “currency” becomes an attribute, the values of which may have conflicting representations in different systems).

3.3 Representing Conversion Knowledge in COIN

Every semantic domain `d` has associated with it a conversion function in the form of a polymorphic method `cvT`, which has the signature:

`d[cvT@d=>d]`.

Hence, if `o1` and `o2` are both instances of `d`, then `o1[cvT@o2->o3]` where `o3` is also an instance of `d` and `o1` and `o3` are *semantically equivalent* in the sense that both have the same interpretation in the domain despite the fact that their print-values may be different. Suppose `o1` and `o2` are instances of `moneyAmt`, and `o1` has value “10,000,000” such that

¹⁰The strategy for oid invention is formally described by the meta-formulae:

`X[M->M(X)] <- X :1 C, C[M=>MC], MC::semanticDomain.`

This oid is of the format `M(X)` where `M` is the method name and `X` is the object to which the method is applied. An object returned in this manner is an instance of the class as defined in the corresponding method signature; i.e.,

`M(X) : MC <- X :1 C, C[M=>MC], MC::semanticDomain.`

(In the above expressions, `X :1 Y` means that `Y` is the *primary class* of `X`. This is easily defined to be the class such that there does not exist an intervening object class `Z` ($Z \neq Y$), whereby `X:Z` and `Z:Y`.)

```

currency(o1)[value->"dem"].  scaleFactor(o1)[value->1]
currency(o2)[value->"usd"].  scaleFactor(o2)[value->1000].

```

The formula `o1[cvt@o2->X]` will bind `X` to an object (with oid `cvt_moneyAmt(o1,o2)`) which has a print-value 6,700 assuming that conversion rate of 0.67 for converting from Deutsch Marks to U.S. Dollars (i.e., $10,000,000[\text{dem} \times 1] = 6,700[\text{usd} \times 1000]$). In the case where a semantic domain `d` is a simple domain (e.g., `number`), `cvt` is vacuous since there are no modifiers: i.e., `o[cvt->o]` is true for every instance `o` of `d`.

```

X[cvt@Y->cvt_moneyAmt(X,Y)] <- X : moneyAmt.
cvt_moneyAmt(_,_) : moneyAmt.
cvt_moneyAmt(X,Y)[value->Value] <- X[cvt(currency)@Y->X1],
    X1[cvt(scaleFactor)@Y->X2], X2[value->Value].
cvt_moneyAmt(X,Y)[M->V] <- not reserved(M), Y[M->V].

X[cvt(currency)@Y->cvt_moneyAmt(currency,X,Y)] <- X:moneyAmt.
cvt_moneyAmt(currency,_,_) : moneyAmt.
cvt_moneyAmt(currency,X,Y)[value->V] <- R:exchangeRate[from->Rf[value->From];
    to->Rt[value->To]], X[currency->Xc[cvt@Rf->_[value->From]]; value->Xv],
    Y[currency->Yc[cvt@Rt->_[value->To]], R[rate->_[value->Rate]],
    prolog(is(V,*(Xv,Rate))).
cvt_moneyAmt(currency,X,Y)[currency->C] <- Y[currency->C].
cvt_moneyAmt(currency,X,Y)[M->V] <- not reserved(M), not M=currency, X[M->V].

X[cvt(scaleFactor)@Y->cvt_moneyAmt(scaleFactor,X,Y)] <- X:moneyAmt.
cvt_moneyAmt(scaleFactor,_,_) : moneyAmt.
cvt_moneyAmt(scaleFactor,X,Y)[value->Value] <-
    X[value->Xv; scaleFactor->_[value->Xf]],
    Y[scaleFactor->_[value->Yf]], prolog(is(Value,*(Xv,(Xf,Yf)))).
cvt_moneyAmt(scaleFactor,X,Y)[scaleFactor->F] <- Y[scaleFactor->F].
cvt_moneyAmt(scaleFactor,X,Y)[M->V] <- not M=scaleFactor, not reserved(M), X[M->V].

```

Figure 5: Conversion functions for `moneyAmt`.

As shown in Figure 5, it is possible to further refine the above scheme of things by defining conversion functions on individual modifiers and allow `cvt` to define how they are to be composed. Hence, the `cvt` method for `moneyAmt` invokes two other conversion functions `cvt(currency)` and `cvt(scaleFactor)` corresponding to modifiers `currency` and `scaleFactor` defined on `moneyAmt`¹¹.

¹¹It is not our intent to populate the domain model with instances from the sources which it purports to integrate, especially since the number of objects in these sources are likely to be very large. Also, F-logic is inherently a logical formalism without provision for updates. The solution we have adopted is to separate the

We raise several other interesting observations concerning conversion functions. A conversion function may require access to information which are not present in the object being operated on. For example, conversion between currencies requires an appropriate exchange rate. In some instances, this knowledge may be encoded as part of the domain model. More frequently, this requires access to some other data sources (e.g., for real-time exchange rates). One option is to identify a source which provides the information as part of the conversion function. A better alternative, shown in Figure 5, is to define the requirement in terms of other entities in the domain model. There are at least two advantages for doing this: first, it allows sources to be independently mapped to the domain model thus achieving our objective of having a loosely-coupled system in which components can evolve independently with minimal impact on the rest; second, it allows source selection to be done dynamically to yield a more optimal query plan, since the information which is needed may be present in different sources which have different access cost.

By virtue of the fact that modifier-values may themselves be semantic values, conversion functions are required to invoke other conversion functions recursively. In the definition of currency conversion, we have been careful to make sure that comparisons between different currencies are mediated by a conversion function (which takes care of different conventions for reporting currencies). It is important to realize that this only states the “intention” and that the actual query plan may or may not include the conversion operation for converting between different currency conventions, depending on whether or not such a conflict does exist. We will revisit this discussion later in Section 4.

3.4 Contexts versus Schemas

In both [21] and [19], contexts are tightly-coupled to an underlying (import or export) schema by adorning each attribute with meta-attributes. The meaning of attributes and meta-attributes in different systems was assumed to be mutually agreed upon even though they were never part of a formal system. This is viable in the earlier discussions since the primary focus was that of achieving semantic interoperability between a (single) source and a (single) receiver, but becomes less realistic when there is a large number of interoperating sources and receivers. We circumvent the above problem by explicitly mapping each data element in an export or import schema to objects in the domain model using an *enriched schema*. All other knowledge pertaining to the interpretation of objects are captured separately in a *context* specification.

invention of an oid from the task of actually assigning it the values corresponding to its methods. As seen from Figure 5, the method `cvt@Y`, when applied to some `X` which is an instance of `moneyAmt`, will return an oid `cvt_moneyAmt(X, Y)`. The print-value of this object is returned by the `value` method and is defined independently (in the subsequent formulae).

The context of a source or receiver encompasses a large class of semantic constraints which are part of the “environment” in which the source or receiver finds itself, and is distinct from the structure of any particular export or import schema. For example, the same organization may maintain several databases, each making available their data via distinct export schemas, but nevertheless have identical assumptions as to how data should be represented and interpreted. In this section, we will limit our discussion to single-layered contexts while highlighting the differences between contexts and schemas through the use of an example. The subsequent sections give concrete illustrations of why this distinction is useful.

```

scaleFactor(P)[value@disclosure->1] <- C:company[profit->P].
convention(Cur)[value@disclosure->"abbrev"] <- Cur:currencyType.
currency(P)[value@disclosure->V] <-
  C:company[profit->P; countryIncorp->_[value->Y]],
  T:country[name->_[cvt@countryIncorp(C)->_[value->Y]];
  officialCurrency->_[cvt@currency(P)->_[value->V]].

```

(a) disclosure context.

```

scaleFactor(P)[value@worldscope->1000] <- C:company[profit->P].
convention(Cur)[value@worldscope->"abbrev"] <- Cur:currencyType.
currency(P)[value@worldscope->"usd"] <- C:company[profit->P].

```

(b) worldscope context.

Figure 6: Sample formulae from the `disclosure` and `worldscope` contexts.

Figure 6 shows the formulae which are part of two contexts called `worldscope` and `disclosure`, which are associated with the databases `worldscope_db` and `disclosure_db` respectively. Notice that all modifiers are assigned values contingent on a context: e.g., company profits are always reported using a `scaleFactor` of 1000’s in the `worldscope` context, but 1’s in the `disclosure` context¹². A more interesting example involves the formula for assigning currency used in `disclosure`: the formula specifies that company profits are reported using the official currency of the country in which a company is incorporated. Hence, when it is necessary to

¹²Previous references to print-values are not parameterized with a context variable, whereas the context assignments formulae in Figure 6 assigns values only to `value@Context`. We get around this easily via the meta-rule:

```
Oid[value->V] <- prolog(=..(Oid, [M,E])), not E:entityType, Oid[context->C; value@C->V].
```

The context which is associated with each modifier (i.e., excluding semantic values which are in fact attributes to some entity) is defined recursively to be that of the semantic value which it modifies, or if it is an attribute, that of the entity:

```
X[context->C] <- not X:entityType, prolog(=..(X, [M,E])), E[context->C].
```

(In the above rule, if `X` is an `entityType`, its context would have been assigned explicitly as shown in Figure 7.)

ascertain what currency is being used in reporting the profit of a company, it is essential to first figure out where the company is incorporated and what currency is being used in that country. In our example, the latter information is not available in `disclosure_db` but has to be obtained from a different data source. As we have mentioned in our discussion on conversion functions, it is not necessary (nor desirable) to make explicit reference to a specific source since we can define it unambiguously with reference to the domain model.

```

company(name(Cname),disclosure_db) : company <- disc(Cname,_,_).
name(X)[value->Cname] <- X:company[source->disclosure_db; keyValue->Cname],
    disc(Cname,_,_).
profit(X)[value->Profit] <- X:company[source->disclosure_db; keyValue->Cname],
    disc(Cname,Profit,_).
countryIncorp(X)[value->CountryIncorp] <-
    X:company[source->disclosure_db; keyValue->Cname],
    disc(Cname,_,CountryIncorp).
% entities exported by disclosure_db has context "disclosure"
X[context->disclosure] <- X:company[source->disclosure_db].

```

(a) Enriched schema for `disclosure_db`.

```

% view definition for user1
imported_disc(N,P,Y) <- C:company[source->user1],
    C[name->_[value->N]; profit->_[value->P]; countryIncorp->_[value->Y]].
% entity referenced by imported_disc
company(K,imported_disc):company <- company(C,disclosure_db):company,
    K=company(C,disclosure_db).
name(C)[value->V] <- C:company[source->imported_disc; key->K],
    K:company[source->disclosure_db; name->_[value->V]].
profit(C)[value->V] <- C:company[source->imported_disc; key->K],
    K:company[source->disclosure_db; profit->_[cvt@profit(C)->_[value->V]]].
countryIncorp(C)[value->V] <- C:company[source->imported_disc; key->K],
    K:company[source->disclosure_db; countryIncorp->_[value->V]].
% entities referenced in imported_disc have context "worldscope"
company(K,imported_disc)[context->worldscope] <- company(K,imported_disc):company.

```

(b) View definition (`imported_disc`) and enriched schema for `user1`.

Figure 7: Enriched schemas for the running example.

As shown in Figure 7, the mapping of a (relational) schema to the domain model is a two-step process. First, the schema identifies one or more entities in the domain model which are being exported by the data source (or imported by a data receiver)¹³. Second, the correspondence between semantic values and attributes are defined, allowing print-values of semantic values to

¹³The oid of entities referenced in the enriched schema are “value-based” in the sense that it is based on the key of the underlying relation(s). The alternative, which is to create a system-wide unique oid for every object known to the system, is prohibitively expensive given our goal of providing large-scale integration.

be identified with attribute values in a schema. The collection of formulae collectively define what data is exported by a source (conversely, what objects are expected by a receiver) and are referred to as the *enriched schema* corresponding to a particular source (or receiver).

4 Query Formulation and Mediation

Figure 8 presents an example that helps to summarize the relationships between the components of a Context Interchange system. This section describes major contributions resulting from the use of this approach. First, the flexibility of the user to query the system in a number of ways (e.g., tightly-coupled, loosely-coupled). Second, the ability of the query planner component of the context mediator to produce a query plan for whatever user query option is chosen.

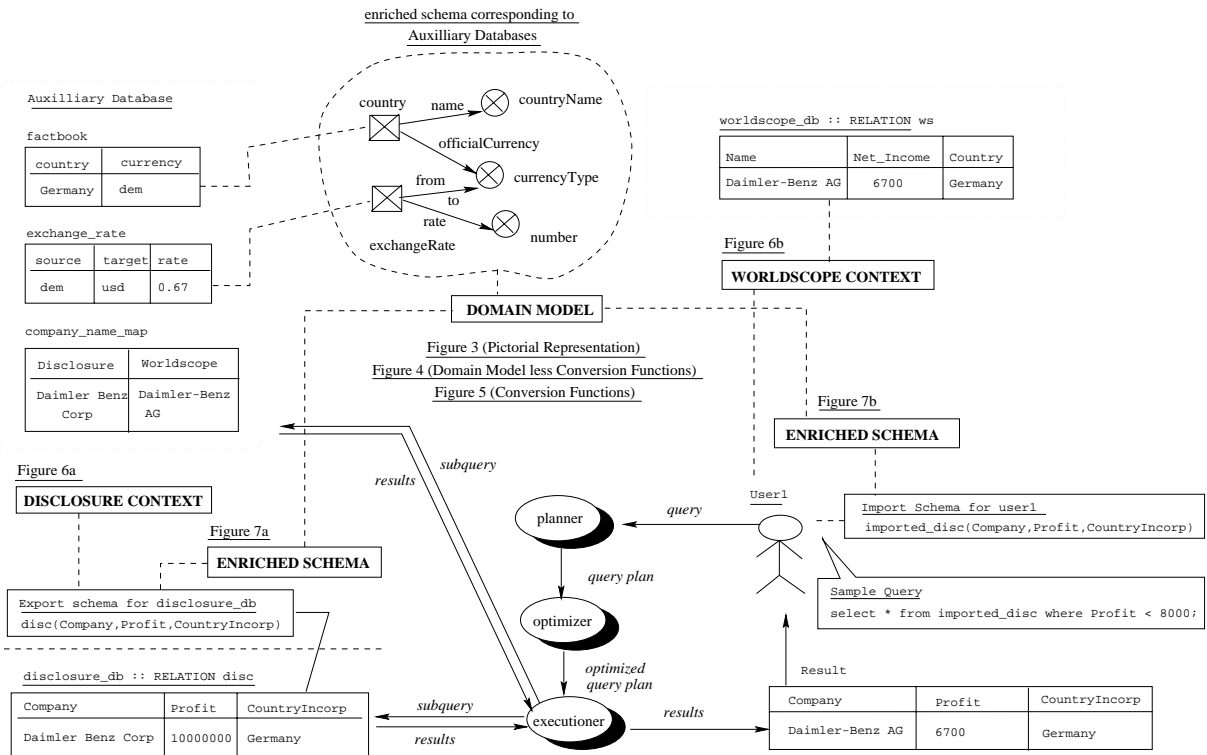


Figure 8: Pictorial summary of running example.

4.1 Query Formulation in a Context Interchange System

There are three different ways by which a query may be formulated in a Context Interchange system: (1) multidatabase queries over the set of export schemas [13]; (2) queries on shared

schemas (defined over selected sources) which presents an integrated view of available data (e.g., as in tightly-coupled systems); and (3) queries on the domain model (or some view of it), stating what information is needed but without specifying what sources to use, leaving the latter decision to the context mediator (this is sometimes referred to as “resource discovery”). It is important to bear in mind that none of the above approaches require the user to explicitly identify or reconcile potential semantic conflicts. Moreover, all of the above options can occur with or without the user having his own context: in the absence of a receiver context, the answer returned will be reported in the context of the source; otherwise, they will be converted to receiver’s context. We will elaborate briefly on each of these.

Multidatabase queries. Multidatabase queries can be supported in a Context Interchange system by (1) presenting users with a set of import schemas, one for each data source, which is structurally identical to the export schemas; and (2) defining each import schema to be a view on the entities exported by the respective source. For example, to allow user1 in Figure 8 to query the export schema of `disclosure_db` directly, we present him with the schema

```
imported_disc(Company,Profit,CountryIncorp)
```

which is defined as a view on `company` instances which are exported by `disclosure_db` (see Figure 7). If we do the same for `worldscope_db`:

```
imported_ws(Name,Net_Income,Country)
```

we could issue the following query

```
select Company from imported_ws w, imported_disc d
where w.Company = d.Name and w.Profit / d.Net_Income > 1.2;
```

which requests the names of companies such that the profit reported in `worldscope_db` has a discrepancy of 20% or more compared to that reported in `disclosure_db`. Notice that unlike Litwin’s MSQL queries in [13], the query does not specify the conversions which must be performed before on company names or profit when comparing these across the two databases. The mediation of semantic conflicts here is automatically taken care by the context mediator. Moreover, the values returned by the query (in this case, company names) will be converted to the context of the user issuing the query.

Queries on Shared Schemas. Instead of providing a user with import schemas which look like export schemas, we can create import schemas which are essentially views defined on export schemas. Returning to the earlier example, suppose we would like to create the shared schema:

```
integrated_view(Name, disc_Profit, ws_Profit)
```

This could be built upon the import schemas defined earlier:

```
define view integrated_view(Name,disc_Profit,ws_Profit) as
select w.Name, d.Profit, w.Net from imported_disc d, imported_ws w
where d.Company=w.Name;
```

Since all attribute values in `imported_disc` and `imported_ws` complies with the `worldscope` context to begin with, so are the values for `integrated_view`. The same query can now be formulated as:

```
select Name from integrated_view where disc_Profit / ws_Profit > 1.2;
```

Queries on the Domain Model. Queries on the domain model (or some view of it) differs from the other two in that sources are not explicitly identified. These are not unlike those requests which are an integral part of conversion functions (e.g., `exchangeRate` in `cvt_moneyAmt(currency,X,Y)`) where source selection is undertaken by the context mediator. Once a source has been identified, query mediation proceeds as before with all semantic conflicts between sources being resolved and data being returned to the receiver in the latter's context.

4.2 Query Mediation

Query mediation in a Context Interchange system takes the form of a top-down evaluation procedure which results in a query plan describing what sources are accessed and what conversion operations must be performed when data are compared or shipped between systems in distinct contexts. We illustrate this procedure with the sample query shown in Figure 8 which, while providing the user with the illusion that the query is directed against the export schema of `disclosure_db`, returns answers in the context of `worldscope`.

On receipt of an SQL query, the context mediator translates this to a conjunctive query which is used as a basis for bootstrapping the evaluation process. For example, the query shown in Figure 8 will be transformed to the following:

```
?- imported_disc(N,P,Y), P<8000.
```

Using the definition of `imported_disc` in its enriched schema (Figure 7), this can be further expanded to

```
?- C:company[source->imported_disc; name->_[value->N]; profit->_[value->P];
countryIncorp->_[value->Y]], P<8000.
```

This transformation is repeatedly applied in a manner similar to classical *SLD-resolution* [14] in which the corresponding logic program consists of the rules present in the domain model, contexts, and enriched schemas of participating sources. Specifically, a subgoal (i.e., a literal in the conjunct) is chosen at each step: if this unifies with the head of some rule, the entire query is rewritten by replacing the subgoal with the body of the rule and applying the appropriate variable substitutions to the result.

Since we are interested in generating a query plan as opposed to using the evaluation process as a basis for generating answers to the query¹⁴, we modify the behavior of the evaluation procedure so that predicates corresponding to relations (in the data source) are not expanded but are instead added to a *query plan*. A consequence of this is that some variables will not be instantiated, which may then cause subgoals which will otherwise succeed to fail: subgoals in this latter category are therefore added to the query plan as well. In both of these cases, the subgoal is deemed to be satisfied and will be removed from the conjunctive goal. On completion, the query plan consists of a conjunctive goal in which all literals are either predicates corresponding to export schemas of some sources, or operations on values returned by them. As an illustration, the query plan corresponding to our sample query is as follows:

```
disc(C1,P1,Y), company_name_map(C1,C), factbook(Y,U),
    exchange_rate(U,"usd",R), P2=P1*R, P=P2*0.001, P<8000
```

In our example thus far, we have assumed that there is exactly one source corresponding to each entity type in the domain model. The presence of multiple redundant sources shows up as multiple formulae with the same entity referenced in the rule head. Hence, if we have two distinct sources *s1* and *s2* for exchange rates, we would have the following formulae (in their respective enriched schemas):

```
currencyExchange(from_to(F,T), s1) : currencyExchange <- r1(F,T,R).
currencyExchange(from_to(F,T), s2) : currencyExchange <- r2(F,T,R).
```

During query mediation, the presence of multiple “applicable” rules leads the mediator to “back-track” on a goal which is previously satisfied. In the absence of further information on what databases actually report exchange rates for what currencies, the resulting query plan will admit both sources resulting in two plans:

```
disc(C1,P1,Y), company_name_map(C1,C), factbook(Y,U), s1(U,"usd",R)
    P2=P1*R, P=P2*0.001, P<8000
```

¹⁴which would be prohibitively expensive since this result in instantiating one tuple at a time from the underlying databases.

and

```
disc(C1,P1,Y), company_name_map(C1,C), factbook(Y,U), s2(U,"usd",R)
P2=P1*R, P=P2*0.001, P<8000
```

One strategy for executing this query is to factor out the common subexpressions and issue subqueries to both `s1` and `s2`, since it is not known which source contains the information which is relevant to our query. If it is known that `s1` and `s2` are replicate of each other, then choosing one over the other would suffice.

The query plan generated by the planner can be further optimized in a number of ways. For example, we could rewrite the last three expressions as $P1 < 8000000 / R$ which may allow a selection constant to be “pushed” into a subselect statement if `R` is known (as when we are only interested in companies from a particular country). Apart from syntactic transformations, the query optimizer will be required to determine the order in which joins should be carried out, and where intermediary results can be shipped to for processing. All of these decisions are contingent on the profile of individual sources and are outside the scope of our logical formalism. Our experiences with optimizing query plans with conversion operations have been reported elsewhere [5] and is beyond the scope of this paper.

5 Context Hierarchies: Inheritance, Defaults, and Non-Montonicity

In addition to the benefits already mentioned, the representation of semantic assumptions in contexts, as distinct from enriched schemas, provides a natural approach to knowledge sharing and reuse. Specifically, contexts can be nested forming a hierarchy which allows an inner context to inherit assertions made in the outer contexts, overriding portions if needed. Formally, we define contexts to be objects with a lineage. For example

```
c0 : context.
c1 : context [parent->c0].
c2 : context [parent->c1].
```

defines `c1` to be a context nested in `c0`, and allows us to infer that `c2` is also nested in `c0` by transitivity. Inheritance and overriding of modifier-value assignments operates on the following rules:

```
X[value@tr(C)->V] <- X[value@C->V].
X[value@tr(C)->V] <- not X[value@C->V], C[parent->P], X[value@tr(P)->V].
```

in which the method `value@tr(C)` recursively traverses the context hierarchy until a value assignment is found for the semantic value `X`. For instance, if we have

```

scaleFactor(X) [value@c0->1] <- X:moneyAmt.
scaleFactor(X) [value@c2->1000] <- X:moneyAmt.

```

and if `m` is an instance of `moneyAmt`, then `scaleFactor(m) [value@tr(c1)->P]` will return `P=1` whereas `scaleFactor(m) [value@tr(c2)->Q]` will return `Q=1000`. Under this scheme, *defaults* are simply values assigned to a “distinguished” context which is the “ancestor”-context for all others (hence, unless they are explicitly overridden, assignments made in this distinguished context are automatically inherited by all others). The primary motivation for having a context hierarchy is to allow context rules to be codified at different levels of abstraction: hence, assumptions common to a group of sources and receivers can be represented at one level, while allowing individual members of this group to further refine these as needed. For example, different divisions within an organization may have idiosyncratic ways of representing certain data, but is also likely to conform to certain organizational practices which are common to all divisions within the organization. The ability to capture contextual assumptions separately from schemas constitutes an important step towards building an information infrastructure which allows inclusion of new systems to become progressively easier as more of the assumptions are being codified.

The overriding mechanism described above can also be easily extended to allow different receivers to define how conflicts are to be mediated by introducing conversion functions tailored to their needs. Suppose we add to the domain model presented earlier a new entity type called `financialReport`, which captures the notion that profitability of a company changes over time and is reported with reference to a given financial year:

```

financialReport [company=>companyName; financialYear=>year; profit=>profit].

```

We could have at least two ways of converting between profit amounts reported in different currencies: we could continue to use a “ball-park” conversion rate (as was done in `cvt(currency)` shown in Figure 5), or we might define a conversion function which takes into account fluctuation of exchange rates over time. The default “ballpark conversion” may be overridden by a receiver in the latter’s context (say, `c1`) using the same approach as is described earlier:

```

cvt_profit(currency,X,Y) [value@c1->V] <-
    % use exchange rate contingent for Year given by X[financialYear->Year].

```

When currency conversion on profit is attempted, this definition will be used in place of the default whenever the receiver’s context is given by `c1`.

In our discussions thus far, we have assumed that, once identified in the domain model, the modifiers of a semantic domain is invariant. This however need not be the case: a user may be indifferent to certain variations in the interpretation of a given semantic domain, or the

variation in itself may be the subject under investigation. With reference to our earlier example, one might be interested to know the names of companies which are identically reported in both `disclosure_db` and `worldscope_db`. This amounts to issuing the query:

```
select Name from imported_disc d, imported_ws w where d.Name=w.Company;
```

If submitted to the context mediator, this query will return the names of all companies common to both databases since conflicting naming conventions is being mediated. An easy way of getting around this is to override the relevant conversion function in the local context to ignore differences in a particular modifier (e.g., `cvt` for `companyName` can be made vacuous, so that `cvt_companyName(convention,X,_) [value->Y]` will always bind `Y` to the value of `X`).

6 Conclusion

We have described in this paper the Context Interchange approach to large-scale semantic interoperability. Our presentation included an in-depth discussion of the COIN data model which provides a firm logical foundation for knowledge representation and reasoning, while taking advantage of various object-oriented features (e.g., encapsulation, hierarchical abstraction, polymorphic methods, and the parsimonious integration of conversion functions as methods). This data model provides the underlying formalism for describing knowledge in the form of the *domain model*, *contexts*, and *enriched schemas* and is a key enabler in allowing sources and receivers to engage in meaning data exchange despite being loosely coupled to one another. The detection and resolution of semantic conflicts is performed by a *context mediator*: the latter uses a top-down evaluation strategy in generating a query plan which describes the conversions which must take place whenever data are being exchanged between systems in different contexts.

Another important contribution of this paper is the distinction made between schemas and contexts which yielded many benefits. Specifically, this allows (1) different sources in a given environment to identify with the same context: resulting in reduced redundancy and making inclusion of new sources progressively easier; (2) queries to be formulated with reference to arbitrary schemas while staying with the same context (which assures them that data returned will comply with expectations codified in the latter); (3) support multiple levels of abstraction and defaults through the use of context hierarchies; and (4) allow conversion functions to be redefined within a given context through the overriding mechanism built into context hierarchies.

The Context Interchange strategy obtains much of its power from the adoption of a very expressive formalism (F-logic), the use of a generic inference strategy (top-down evaluation), and a lazy-evaluation approach which defers the detection and reconciliation of conflicts as much as possible. In the extreme, conflict mediation is performed only at run-time, when a query is

submitted. This however need not be the only model: it is conceivable that plans generated for different query “templates” can be cached in a receiver’s system. This plan describes unambiguously what conversion and database operations must be performed to satisfy a query, and is not unlike the view defined under the tight-coupling approach. However, unlike classical shared schemas, there is no need to manually update our “view definition” each time there is a change in the system: all that is needed is for the context mediator to generate a new plan using the updated information.

It is worthwhile noting that the adoption of an object-oriented data model for database integration is not novel in itself. Pegasus [1] is one such example and many others have been documented [3]. However, to the best of our knowledge, all of these proposals retain the full-flavor of tight-coupling systems: the benefits of object-orientation are accrued solely towards facilitating the construction of view mappings to a global schema of some sort.

There are of course other projects which share our aspirations in combining the benefits of loose- and tight-coupling systems. For example, TSIMMIS [17] approaches this by requiring each data source to export objects in some “self-describing” format and has avoided any reference to a shared domain model but provides “man”-pages describing the semantics of objects being exported. The fact that these descriptions are written in natural language suggests that they are not suitable for automated conflict detection and resolution. Our work also bears some relationship to McCarthy’s attempts at defining a general theory for *contexts* [16]. In particular, Faquhar et al. [6] has sought to apply McCarthy’s framework to the Context Interchange ideas we have articulated elsewhere, but that work has not yet paid as much attention to issues of scalability, nor provided an emphasis on user flexibility, and query plan generation.

A fair amount of ideas presented in this paper has been realized in a proof-of-concept implementation on the WWW which provides for integrated access to both traditional databases and non-traditional sources (e.g., web pages). In addition to this effort, we are also investigating a number of open problems which stem from our framework; these include issues related to query optimization, and in particular, source selection in situations where data may be replicated or fragmented (horizontally and vertically).

Acknowledgment

We are indebted to Ed Sciore and Arnie Rosenthal for their comments on previous drafts of this paper.

References

- [1] AHMED, R., SMEDT, P. D., DU, W., KENT, W., KETABCHI, M. A., LITWIN, W. A., RAFFI, A., AND SHAN, M.-C. The Pegasus heterogeneous multidatabase system. *IEEE Computer* 24, 12 (1991), 19–27.
- [2] BRIGHT, M., HURSON, A., AND PAKZAD, S. A taxonomy and current issues in multidatabase systems. *IEEE Computer* 25, 3 (1992), 50–60.
- [3] BUKHRES, O. A., AND ELMAGARMID, A. K., Eds. *Object-Oriented Multidatabase Systems: A Solution for Advanced Applications*. Prentice-Hall, 1996.
- [4] CLARK, K. Negation as failure. In *Logic and Data Bases*, H. Gallaire and J. Minker, Eds. Plenum Press, 1978, pp. 292–322.
- [5] DARUWALA, A., GOH, C. H., HOFMEISTER, S., HUSSEIN, K., MADNICK, S., AND SIEGEL, M. The context interchange network prototype. In *Proc of the IFIP WG2.6 Sixth Working Conference on Database Semantics (DS-6)* (Atlanta, GA, May 10 – Jun 2 1995). To appear in LNCS (Springer-Verlag).
- [6] FAQUHAR, A., DAPPERT, A., FIKES, R., AND PRATT, W. Integrating information sources using context logic. In *AAAI-95 Spring Symposium on Information Gathering from Distributed Heterogeneous Environments* (1995). To appear.
- [7] GOH, C. H., MADNICK, S. E., AND SIEGEL, M. D. Context interchange: overcoming the challenges of large-scale interoperable database systems in a dynamic environment. In *Proceedings of the Third International Conference on Information and Knowledge Management* (Gaithersburg, MD, Nov 29–Dec 1 1994), pp. 337–346.
- [8] GRUBER, T. R. The role of common ontology in achieving sharable, reusable knowledge bases. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 2nd International Conference* (Cambridge, MA, 1991), J. A. Allen, R. Files, and E. Sandewall, Eds., Morgan Kaufmann, pp. 601–602.
- [9] HULL, R., AND KING, R. Semantic database modeling: survey, applications, and research issues. *ACM Computing Surveys* 19, 3 (1987), 201–260.
- [10] HURSON, A. R., BRIGHT, M. W., AND PAKZAD, S. H. *Multidatabase Systems: an advanced solution for global information sharing*. IEEE Computer Society Press, 1994.
- [11] KIFER, M., LAUSEN, G., AND WU, J. Logical foundations of object-oriented and frame-based languages. *JACM* 4 (1995), 741–843.
- [12] LANDERS, T., AND ROSENBERG, R. An overview of Multibase. In *Proceedings 2nd International Symposium for Distributed Databases* (1982), pp. 153–183.
- [13] LITWIN, W., AND ABDELLATIF, A. An overview of the multi-database manipulation language MDSL. *Proceedings of the IEEE* 75, 5 (1987), 621–632.

- [14] LLOYD, J. W. *Foundations of logic programming*, 2nd, extended ed. Springer-Verlag, 1987.
- [15] LLOYD, J. W., AND TOPOR, R. W. Making prolog more expressive. *Journal of Logic Programming* 1, 3 (1984), 225–240.
- [16] MCCARTHY, J. Notes on formalizing context. In *Proceedings 13th IJCAI* (1993).
- [17] PAPA-KONSTANTINOY, Y., GARCIA-MOLINA, H., AND WIDOM, J. Object exchange across heterogeneous information sources. In *Proc IEEE International Conference on Data Engineering* (March 1995). electronic version at <http://www-db.stanford.edu/tsimmis/publications.html>.
- [18] SCHEUERMANN, P., YU, C., ELMAGARMID, A., GARCIA-MOLINA, H., MANOLA, F., MCLEOD, D., ROSENTHAL, A., AND TEMPLETON, M. Report on the workshop on heterogeneous database systems. *ACM SIGMOD RECORD* 19, 4 (Dec 1990), 23–31. Held at Northwestern University, Evanston, Illinois, Dec 11–13, 1989. Sponsored by NSF.
- [19] SCIORE, E., SIEGEL, M., AND ROSENTHAL, A. Using semantic values to facilitate interoperability among heterogeneous information systems. *ACM Transactions on Database Systems* 19, 2 (June 1994), 254–290.
- [20] SHETH, A. P., AND LARSON, J. A. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys* 22, 3 (1990), 183–236.
- [21] SIEGEL, M., AND MADNICK, S. A metadata approach to solving semantic conflicts. In *Proceedings of the 17th International Conference on Very Large Data Bases* (1991), pp. 133–145.