Articulation Management for Intelligent Integration of Information

David A. Maluf, Ph.D. Peter B. Tran

Abstract—

When combining data from distinct sources, there is a need to share meta-data and other knowledge about various source domains. Due to semantic inconsistencies and heterogeneity of representations, problems arise in combining multiple domains when the domains are merged. The knowledge that is irrelevant to the task of interoperation will be included, making the result unnecessarily complex.

This heterogeneity problem can be eliminated by *mediating* the conflicts and managing the *intersections* of the domains. For interoperation and intelligent access to heterogeneous information, the focus is on the intersection of the knowledge, since intersection will define the required *articulation rules*.

An *algebra over domain* has been proposed to use articulation rules to support disciplined manipulation of domain knowledge resources. The objective of a domain algebra is to provide the capability for interrogating many domain knowledge resources, which are *largely semantically disjoint*. The algebra supports formally the tasks of selecting, combining, extending, specializing, and modifying components from a diverse set of domains.

This paper presents a domain algebra and demonstrates the use of *articulation rules* to link *declarative interfaces* for Internet and enterprise applications. In particular, it discusses the articulation implementation as part of a *production system* capable of operating over the domain described by the *IDL* (interface description language) of objects registered in multiple *CORBA* servers.

Keywords—

Intelligent Information Systems, Domain Algebra, Articulation Rules, Domain Algebra, Semantic Interoperation, Heterogeneous Systems, Integration

I. Introduction

Today, designers, developers, and users realize that information in private and public databases, as on the Internet, provides increasing opportunities to enhance productivity. Understanding the content of the available information requires the use of domain knowledge. For example; acquiring an air fare from a travel agency on the Internet requires an understanding of layout (e.g. parsing) of the corresponding HTML pages. Furthermore, the effective use of the knowledge to support problem solving also requires the use of multiple domain resources, for instance comparing prices from different agencies to meet a user query.

A number of technologies have been developed to support large-scale interoperation among distributed applications [41]. However, managing large-scale interoperation of domains remains a task which requires many levels of expertise and an adherence to standards.

Many existing systems have strong notions of interfaces. Efforts like eXtensible Markup Language, the next generation of HTML, are only ways to put structure into a web page. These interfaces allow the specification of the domain's *knowledge* or the domain component syntax. Leading efforts in interoperating among multiple domains are often implemented as the union of multiple domains. An immediate increase in data integration is noticeable to data integration architects. Yet, there are consequences [1] to the union of multiple domains which turn out not to be efficient for the evolving needs of customers, such as resource allocation (storage and staffing). Worst, however, the union will become impossible to maintain.

For interoperation and intelligent access to heterogeneous information, the focus should be on the intersection of the knowledge, since intersection will define the required articulations. The term *articulation* refers to the linkages which join concepts across domains [15]. The emergent need to define articulations between data resources has been demonstrated and described in [24][47]. For instance, *Yahoo*'s hierarchal classification and *Auto.com*, "Yahoo's category Business-and-Economy/Companies/Automotive" and "Auto.com daily reviews" can be well articulated (linked).

We extend and generalize the identification of the articulation to a set of manipulations, such as selecting, combining, extending, specializing, and modifying components from diverse, common and domain-specific knowledge. To deal with most of these issues, a domain algebra has been proposed in [46] which is intended to support disciplined manipulation of knowledge resources. The representation of vocabularies and their structure is termed *domain knowledge* whereas the operations that combine and partition the domain knowledge in a sound and well-behaved manner are termed a *domain algebra*. The basic algebra consists of three operations, namely intersection, union and difference (negation is considered an alternate form of the difference). *Knowledge* in this paper is limited to the knowledge that an expert can extract from a domain and not the domain itself [2].

The objective of a domain algebra is to provide the capability for interrogating many knowledge resources, which are largely semantically disjoint, but where *articulations* have been established. Articulation rules will enable on their own a new type of interoperability from *Server-to-Server*.

This paper describes the role of a domain algebra in a *mediated architecture* among declarative interfaces. It also demonstrates the use of an algebra which provides users and system developers with the ability to intelligently manipulate components in real time.

The idea of combining articulation-rules [37] with declarative interfaces is complementary: declarative interfaces are primarily about specifying component syntax and distributed implementations [39], whereas articulation rules are a promising research outcome recycled from Artificial Intelligence and have addressed in the past issues of component design, component binding, and component semantics.

A. Background

An introduction to *domain algebra* is presented in references [46][47]. In these papers, the advantage of a domain algebra is described. Some suggested interoperation functionalities of the domain algebra are presented in [28]. Also, there has been a significant amount of research in the interoperable systems community. The representative literature in semantic interoperations is presented in [41][13][12]. Much of this work conforms to describing interfaces which mirror the effort in the database community [10][1] that addresses the problem at the schema integration level [25][48]. There are a number of similarities with the database and knowledge-base community and the proposed work: the concept of articulation [15][24], translating heterogeneous information into a meta-level model [34][45], active

Computational Sciences Division, NASA Ames Research Center, Mail Stop 269-2, Moffett Field, CA 94035. E-mail: maluf@email.arc.nasa.gov

QSS Group Inc., Mail Stop 269-4, Moffett Field, CA 94035. E-mail: pbtran@mail.arc.nasa.gov

[1] Also down-loading Internet classifications are copyright infringement by U.S. Law.

[2] e.g. complete schema dump.

database [43] and associating constraints and triggers with objects [17]. This paper focuses on the adaptation of methods from the heterogeneous database literature, mediation and integration aspects to the problem of disciplined manipulation of information sources across networks, languages and platforms (e.g. HTML, XML, CORBA etc.).

B. Object Oriented Bindings

Many existing systems have strong notions of interfaces. These interfaces allow binding to services provided by their corresponding architectures if their interfaces are described. New standard efforts like eXtensible Markup Language allow a Document Type Definition (DTD) where grammars and structures of the markup language are defined [49].

Systems like Object Request Broker (ORB) [38] and Inter-Language Unification (ILU)[16] promote interoperability via interfaces between domains. Domains are known by their interfaces and became an industry standard with the Common Object Request Broker Architecture (CORBA) often known as *The CORBA Standard*. In an object-oriented approach, a domain's interface is bound to the system's object-oriented interface definitions language. This paper will illustrate the implementation of an ORB-to-ORB interopertion and hence demonstrate a new and modern approach to interoperability from Server-to-Server [3].

C. A modern Approach: Server-to-Server

Considering that declarative interfaces are primarily about specifying syntax and component implementation, it appears that our approach complements the concept of declarative interfaces for interoperabilty problems (e.g. interoperability aspects in XML and CORBA). *There are two main differences with the current technology: we support (i), the heterogeneity of the interfaces, and (ii) the autonomy of the interfaces. The first problem relates to the problem of semantic mismatch and granularity incompatibility. The second problem is that interfaces are defined by resources available at a compile time and hence they may not fully cooperate in runtime* (e.g a web page update). Some issues similar to these problems have been discussed with respect to the modern concept of coordinating distributed objects in declarative interfaces as in [21][36].

The problem of interoperation among heterogeneous systems is central to the area of integration, as represented in [50]. The proposed modern approach of Server-to-Server has been adopted by *Science Gate Bay* [42] and has been extended to support information integration on the Internet with XML. Some of the work has appeared in parts under [28], [27]. Some early implementation aspects have appeared as a public software release [29].

II. Knowledge Representations and Interoperation

The development of the *mediation* model reported in this paper is motivated by the need of interoperability among existing domain-specific representation of knowledge (structure and layout) and their respected formalisms (HTML, XML, Objects etc.).

Although the spirit of this paper is to underline practical aspects recycled from Artificial Intelligence with an objective of matching industry current needs, what follows is a brief review of what is commonly known in the Knowledge Representation and Artificial Intelligence community.

The series of knowledge representation formalisms and frameworks starting with KL-One [7] and currently culminating in systems like Classic [5] and LOOM [35] provide powerful tools and knowledge expressiveness. However, they were not intended to interoperate. How much has to be added to their infrastructure and reasoning capability to achieve knowledge interoperability is still unclear.

While knowledge representation is thought of as being a way to resolve integration problems, most knowledge representation formalisms have focused on paradigms which assume an integrated environment and have been careless about managing the exceptions. In our approach, we focus on these exceptions.

From a research and technical point of view, there have been two recent efforts that open up possibilities for meaningful knowledge interoperation: the development of context logic [31] and knowledge interfaces for sharing [37]. The advance in context logic is the notion of translating encoded knowledge relative to its context and hence relates the knowledge to its domain. This is the approach taken in the re-engineering of Cyc [30] where micro-theories confine the contextual differences [24]. Advances in knowledge sharing revolve around translating multiple knowledge, from one formalism, to multiple formalisms. However, the problem of translating many domains into different representations will create several problems. Semantic inconsistencies will arise from the terms and relationships used from the merged domains. Additional inconsistencies occur when the knowledge-content differs both in semantics and in compositional granularity. In addition, the union of multiple domain knowledge includes irrelevant knowledge and the result will be large, unorganized, and disproportionally costly to process.

An early formal paradigm in the direction of porting knowledge from one representation language into multiple ones was done by *Ontolingua* [23]. Ontolingua is a mechanism for translating from a standard syntax into multiple-representation systems. However, directly translating entire knowledge to any arbitrary representation leads to irrelevant knowledge and semantic inconsistencies, disproportioned in content.

Interoperation became an industry fact with CORBA. CORBA is a system of standards and specifications that describes how software components, as being the domain knowledge, can interoperate across networks, languages and platforms. CORBA allows for client-server interaction between heterogeneous objects distributed over a wide-area network. CORBA makes meta information describing the objects in a system and their interfaces available so that it can access other objects. Any object connected to an Object Request Broker (ORB) can play simultaneously the role of client and server and hence Objects can initiate calls and respond to requests. ORB is the part of CORBA which facilitates client-server communication and interaction between distributed objects. To reach object interoperability and for objects to plug and play, clients have to know exactly what they can expect from every object they might call upon for a service.

With the success of Hypertext Markup Language (HTML) and large-scale content distribution of heterogeneous information, industry pushed the technology further with the eXtensible Markup Language (XML). XML was primarily intended to meet the requirements of large-scale Web content providers for industry-specific markup, vendor-neutral data exchange, one-on-one marketing, workflow management, the processing of Web documents by *intelligent clients*, and most meta-data applications.

A. Engineering a Functional Model

Information integration for Internet and enterprise application is a relatively new but growing area of concern, moving

[3] compare to client-server

well beyond objects, Hypertext and the sharing of documents. Software and data provide the capability for active sharing of concepts and many other tasks beyond the ubiquitous and overloaded information world. The Internet proved to be practical and functional fact and deserves to be looked at pragmatically. Even though this paper is driven by a long-range vision, the proposed engineering methodologies allow a continuous stream of early experiments and development. Such an approach is an attractive alternative to the more philosophical approach, which assumes that all fundamental issues have been solved first.

In this paper, we plan to manipulate information labeled with the domain context where context will encapsulate the meaning. Another objective of this paper is to support knowledge abstraction between different knowledge compositions. Abstraction with context will hence play an essential role in the algebra since different domain compositions have different context granularities. It is yet another objective of this paper to delineate the extent of domain knowledge into partitions. But the main objective of this paper is to assert articulation rules which link domain knowledge. Hence, the planned domain algebra should bring about better knowledge scalability.

The process of abstraction separates the domain from its implementation. For instance in articulating Yahoo's category "Business-and-Economy/Companies/Automotive", "Auto.com" would not require the inherented hierarchy (i.e. /Business-and-Economy/Companies/). The notion of separation was initially suggested in the scheme of the Agent Communication Language [21]. We realize that there is much leverage in articulating the domains in the mediation level within partitions because that is where the abstraction is best reasoned about and understood.

When designing a domain algebra for diverse domains, it is important to understand the constraints that the articulation rules set over the underlying knowledge. Articulating a domain at multiple levels increases the domain scope when compared to the articulation rules set only at the atomic knowledge in the domain. This is the case when respectively articulating tables and attributes in relational databases and articulating only the attributes. Writing articulation rules requires a good understanding of the domain as well as an established expertise.

Our hypothesis is that a domain algebra is feasible within a mediated architecture. Furthermore, when the articulation rules establish a rule-based environment, we sustain the operations needed by the algebra. The mediator will seamlessly re-partition and re-combine articulation rules.

B. Mediation

A *mediated model* scales and partitions domain knowledge using the articulated knowledge rather than the domain knowledge itself. Every integration goal requires new objective domain expertise to hand-craft or statistically learn (e.g. data mining) [8][3][26] the necessary articulation rules. In a practical sense there is no magic in current leading Intenet technology such as "NetNanny" [33] or "Alexa" [4] where both technologies directly articulate the Internet based on IP numbers [4].

While an information mediation admits formal definitions in the literature, mediation obeys strong engineering practices. To build a functional application, the focus should be on the intersection of the knowledge among the domains since intersection will define the required *articulation rules*. Articulating domains introduces a radical change in the definition of the domain expert.

An additional new role while articulating domain knowledge for the domain expert is the *partitioning* of the *articulations*. Partitioning separates the articulation rules into *partitions* or bundles (modules). For example the articulations between domains "Yahoo's category Business-and-Economy/Companies/Automotive" and "auto.com" reside in one partition whereas the articulations between "auto.com" and "General Motors (gm.com)" reside in another partition. Practically one may be better off with additional partitions for each pair of knowledge sources to enhance potential sharing (e.g. generic rules like lower/upper case mapping conventions).

It is clear in this model that partitions and contexts are distinct. Partitions are regarded as a "bundle" of articulation rules among domains. On the other hand, contexts establish the labels over the domain knowledge. Since partitions are more likely to be encapsulated, redundancy in the domain knowledge and contexts occurs when domains tend to be homogeneous. The formulated articulation rules are used for linkages across domains will also exhibit redundancy, for instance, articulating pairwise airline companies with airline companies. However, what would be preferred and more meaningful would be to articulate pairwise airline companies with travel agencies. With a little care and hand-crafting of the partitions, redundant articulation would be shared by sharing the appropriate partitions (e.g. word stemming, names mapping, special parsing etc.). In general our approach will scale much better in heterogeneous domains.

C. Declarative Interfaces

In this section we illustrate an example of a domain knowledge written in CORBA's specific *Interface Definition Language* [38]. The same example will be used in the implementation section to demonstrate the interoperability.

To reach object interoperability and for objects to plug and play, clients have to know exactly what they can expect from every object they might call upon for a service. In CORBA, the services that an object provides are described to interface between the object itself and the rest of the system. The objective of the interface is twofold: (i) it informs clients of the services that the objects provide as well as the access method to invoke these services and (ii) it informs the communications infrastructure of the format and syntax of the access methods.

CORBA *Interface Definition Language* (IDL) is defined as a language for describing the interfaces of software objects. An interface is a description of the set of possible operations that a client may request of an object [38]. An IDL interface specification contains declarations of types, exceptions and constants. As most declarative interfaces, IDL is independent of programming languages, and is used to describe objects implemented. For instance, the IDL specification of an object `Bicycle` and class `Shop` can be described as follows:

```
interface Shop {
   readonly attribute long parent;
   void set(int long value);
   long get();
};
interface Bicycle : Shop {
   readonly attribute short size;
   readonly attribute short color;
   short getSize();
   short getColor();
};
```

[4] "Alexa" establishes mappings on IP addresses and display the mappings when an IP is selected. While "Alexa"'s rules are statically learned, "NetNanny"'s rules are hand-crafted into a boolean type block/allow.

The information represented by the IDL specification for any objects connected to a CORBA server is compiled and stored in the *Interface Repository* service which the server provides. The interface repository can be examined by objects on the server in order to ascertain what other objects are connected to the server and what interfaces they provide. This allows an object to request services from other objects on the server without having prior knowledge of the other objects or their interfaces.

III. Terminology, Definitions and Assumptions

The domain algebra scales partitioned domain knowledge by operating over the articulation rules. The partitioning of a domain knowledge is virtually treated by First Order Logic (FOL) formalism since the first order logic is explicitly treating the articulation rules themselves. This traces back early work on algebras [14] which were demonstrated by first order predicate calculus.

Exporting data independently of the specific implementation defines the interoperability capability of a domain. The domain knowledge that describes the data has no representation and is only defined by the terms and relationships. For example, data could reflect the price of a component whereas the knowledge is the name of the component. This decomposition of a domain to its knowledge enables pattern matching over the terms and relationships.

In this paper, we also address the problem of how to abstract and encapsulate encoded knowledge within contexts. Knowledge abstraction as used in this paper composes declarative knowledge compositions, keeping their context through formal predication. We also address the foundation of context formulation as a basic and simple problem in propositional calculus. Establishing context is also a novelty of this paper, and is a topic which is poorly discussed in the literature. The abstraction and context transformations represent the needed knowledge from the domain source model.

The mediation model produces the environment needed to provide users and system developers with the ability of an algebra to perform the tasks of selecting, combining, extending, specializing, and modifying multiple domain-specific knowledge. These manipulations will support the interoperation of descriptions of topics of interest when using the knowledge. These descriptions are reusable by multiple applications that need to access diverse data sources. The descriptive formalism makes the mediated architecture maintainable in rapidly changing environments.

Domain Knowledge: Predicates are the basic construct of declarative knowledge. For example, one can express in first order logic the fact that a `Book` is above the `Table` by taking a relation symbol such as `Above` and defining a predicate `Above(x,y)`. Hence, for the object symbols `Book` and `Table` we can declare the proposition `Above(Book, Table)` [20]. Often, a predicate contains semantic conjunctions or disjunctions within its syntax [18] to express complex relation constructs.

To deal with the inadequacy of semantics and to conform to the syntactics of predicate logic, structured predicates are separated into simpler atomic propositions. If for example a domain considers the proposition `Above(Book` *and* `Pen,Table)`, it is equivalent to the conjunction of the following two predicates `Above(Book, Table)` $\wedge$ `Above(Pen, Table)`. In general, predicates are atomic and do not contain semantic operators.

Writing the domain knowledge is a task that often is automated by wrappers. Either way, expertise is required in writing the knowledge directly or formulating the wrapper itself. □

Abstraction: Abstraction is equivalent to the production of simpler approximations of the domain knowledge often driven by approximation rules (e.g. stemming rules). When domain knowledge involves a large vocabulary, abstraction is also the process of aggregating the domain model to another involving smaller vocabulary and fewer constant, as in the instance of using the term "Automotive" instead of "Business-and-Economy/Companies/Automotive". Often the aggregation is performed by translating the declarative knowledge predicates and grouping the vocabulary and constants into arbitrary well-formed formulas. In [2], one can distinguish the different types of abstraction such as qualitative abstraction, quantitative abstraction, terminological abstraction and temporal abstraction. In our work, however, the notion of abstraction focuses on manipulating knowledge within context.

For example, in the case of applying the relation symbol `Above` to the objects `Book` and `Table` and a third argument denoting a situation s, say {`Llibrary, Office, Home`}. Abstraction in granularity is achieved when the proposition `Above(Book,Table,`s`)` is translated into `Above(Book,Table)` given the context `(s)`.

Abstraction is also a task that is automated by wrappers. Text clustering is a currently used as an abstraction technique where distances (arbitrary metrics) to a given database of collection of key terms (i.e. contexts) are computed [9]. Either way expertise is required in writing the axioms or formulating the wrapper itself. □

Context: Context has been proposed as a means of defining the validity of a sentence relative to a situation. Formalizing contexts [31][24][11] allow predicates for fixed situations to be "lifted" to more dynamic contexts where situations change. The context formalism is an extension to first-order logic in which sentences are valid within a context. To this end, we use the denotation of $\texttt{ist}(c; p)$ such that we have a formula of a proposition p which is true in a context c. For example given a context `Office`, one can write `ist(Office; Above(Book, Table))`. In this paper we drop `ist` and consider a concise and simple form as in the formulas $(\texttt{c}; \texttt{p})$. From the previous example we may write the proposition `(Office; Above(Book,Table))`. In the implementation of the latter example, the mediator was programmed with a pattern matching template of the form `(AXIOM:123 (CONTEXT Office) (RELATION Above) (OBJECT Book, Table))`.

At this point it is worth noting a difference between the context logic formalism approach and the one in this paper which is that context logic defines a default coreference rule which states, that as a default, the meaning of a symbol does not change from one context to another. We consider that symbols never mean the same. The key in resolving ambiguity of meanings of a term is in observing the corresponding contexts.

Hence, for the interface example above for the context `Factory` for the bicycle components, one can write

$$\texttt{ist}(\texttt{Factory}; \texttt{Spoke}(\texttt{x}) \wedge \texttt{Wheel}(\texttt{y}) \wedge \texttt{Frame}(\texttt{z})).$$

□

Articulation Rules: The term *articulation axioms* has been established in [24] but refers to the rules that are used for translating concepts across domains. In this paper we refer to these rules as the *articulation rules*. For instance, when the bicycle components of a `Factory` match the concept of `Bicycle` in a `Shop`. Hence one can write

$$\texttt{ist}(\texttt{Factory}; \texttt{Spoke}(\texttt{x}) \wedge \texttt{Wheel}(\texttt{y}) \wedge \texttt{Frame}(\texttt{z}))$$
$$\Leftrightarrow$$

$$\texttt{ist}(\texttt{Shop};\texttt{Bicycle}(\texttt{z}))$$

The articulation rules and their specification are the components of the domain algebra which describes the linkages that handle interoperation between the independent systems. The articulations allow equivalence from server-to-server and the interactions between heterogeneous objects distributed over a wide-area network. □

Partitioning: In a certain application, there is an interest in certain *Articulation Rules* rather than the complete intersection of the domain. Partitioning is equivalent to the production of subsets of the articulation rules. When involving multiple or large domains, partitioning is the process of selecting, combining, extending, and specializing sets of articulations. For example a domain could be abstracted automatically in one partition whereas an expert assertions on the domain are maintained in another. For instance, a domain expert of a `Factory` and `Shop` models can group

$\texttt{Partition}(\texttt{SEASONAL MARKET})$:

$$\texttt{ist}(\texttt{Factory};\texttt{Spoke}(\texttt{x}) \wedge \texttt{Wheel}(\texttt{y}) \wedge \texttt{Frame}(\texttt{z})) \Leftrightarrow \texttt{ist}(\texttt{Shop};\texttt{Bicycle}(\texttt{z}))$$

Where the partition `SEASONAL MARKET` specializes an articulation rule. □

IV. A Domain Algebra

Automatic reasoning about the interfaces requires a formal approach to the transformation and manipulation of the articulation rules. Hence a set of operations are established for the needed manipulations. These operations describe the domain algebra.

The domain algebra is symbolically composed of two types, namely partitions, articulation rules and operation symbols. The articulation rules are atomic elements of the partitions. The Partitions and articulation rules themselves are the elements of the algebra. On the other hand, the symbols of operations, such as $\bigcap$, $\bigcup$ and $-$, stand for the algebra operations. For multiple domain sources, the complete operations among domains are:

Operation	Symbol	Semantics
Intersection	$\bigcap$	Create sharable expressions
Union	$\bigcup$	Create all expressions
Difference	$-$	Create not shared expressions

- *Intersection:* The intersection is the first concept of the domain algebra since it allows the algebra to bring together two domains. It is equivalent to an `AND` operator. The concept of intersection is not exactly like the predecessor algebras: the intersection is hand-crafted and reflects the articulation rules.
- *Union:* The union concept allows the algebra to bring together two domains to form a new one. It is equivalent to an `OR` operator. However the algebra lacks a formal approach to eliminate redundant knowledge that is common to both. This leads to several ways of establishing the unions of multiple domains. It is convenient to think of knowledge as not being redundant if not explicitly specified by the articulation rules. Similarly to the natural join in relational databases, the domain algebra union joins interfaces when they link through shared articulation rules. The union is restricted only to the knowledge that the rules relate to. In object oriented models, inheritance and class ownership are typical relations that an articulation rule can relate to.
- *Difference:* The difference concept completes the algebra and its presence compensates for the absence of negation [46]. The difference operation retrieves the elements in domains that are `NOT` covered by another. Hence, the difference operation results in asymmetrical results and is not commutative.

Such an algebra can provide a basis of interrogating multiple interfaces which are semantically disjoint, but where a shared knowledge base has been established.

V. Surrogates: Knowledge Formulation and Rewriting

The previous sections defined a few requirements of the mediated architecture. However, these requirements place no restrictions on the possible context entailment propositions can have. In this section, we explore the effect of the partitioning against the effect the domain context for semantic distinctions.

In this section we also explore the implication of the mediated architecture on using the domain knowledge. We assume that the underlying knowledge has been compiled into declarative languages which correctly conceptualized the domain knowledge as propositions. The mediator addresses several tasks in using the domain knowledge, namely resolving implementation differences, interpretation and partial information.

1. *Resolving composition differences:* As some of the work on designing interfaces focuses on designing data models, one can realize the different possibilities for modeling concepts. For an interoperability problem such as in data integration process, one should focus on relating the difference among data models, e.g., mapping the relational model to the object model which requires structural knowledge [44]. However, even if we consider for example only databases using the same data model, there are significant differences which make the task of relating the semantics of the data model difficult. These differences are due to their schema composition.
2. *Interpretation:* To permit the explicit knowledge as in the case of databases to interoperate with other sources, it is not sufficient to simply use the information on the basis of vocabulary used in the domain. The vocabulary used does not correspond to intended meanings. For instance terms like `Id`, `SSN` etc. should map to `Identification`, `Social Security Number` respectively. For each of these domain schemas, their corresponding knowledge is examined in parsing their vocabulary and the specification of their relationships, new surrogated (aliases) terms can be used. Interoperability can occur in a sound manner with propositions.
3. *Partial information:* To handle partial information extracted from declarative interfaces. This problem in database interoperability is simply typified by the symptoms of most directed graphs which is their inability to handle partial information [20]. For example it is very difficult to assert a proposition in a object hierarchy without a reference to a root object. The lack of reference in general is often found with systems that lack external schemas. Similar partial information populates most semistructured information systems, e.g., the World Wide Web.

A. Knowledge Formulation and Rewriting

Our interest is in expressing articulation rules in the framework of propositional calculus. This interest is based on a combination of two features of the mediated architecture. First the knowledge needed can be expressed in a form more or less independent of the uses to which the knowledge might be part of. Second the reasoning performed by the partitioning process involves basic but simple logical operations on these propositions.

In the implementation of the mediator, we specify the proposition rewriting within first-order logic and use the formalism to constrain the propositions in terms of their context. A context can be thought of as a set of terms labeling a set of propositions. Intuitively, we assume a context production rule which states that the meaning of a proposition admits the context defined by the symbols stated within the proposition. For example, the proposition `Above(Book, Table)` has as possible contexts `Book` and `Table`. Although the definition of the context production rule is not very suggestive, it is not the case when considered within the framework of propositional calculus and context logic. In general, we consider the formulas as propositions of the form

$$(c_1, ..., c_M; p_1, ..., p_N) \tag{1}$$

which are to be taken that the propositions $p_1, ..., p_N$ are true in the contexts $c_1, ..., c_M$. For example, if we consider the proposition `(Office, Book; Above(x, Table))`, then we know that predicates `Above(x, Table)` is true in the context of `Office` and `Book`. The aim of reformulation context is not to use deduction as the computational framework, but rather to integrate knowledge into optimal articulation rules when the interoperation objectives are clear.

One can become concerned with the number of possible propositions that can be calculated from Equation 1. We simplify the problem by focusing only on the articulation needed for interoperation. Since automated inferences are potentially capable of processing the symbolic propositions, the need for rules about how to process the knowledge becomes essential. Although there are no general rules in establishing the rewriting of the propositions, the mediated architecture supports the two performative rules: *spanning context* and *specializing*. Both reduce the scope of the interoperation.

1. *Spanning Context:* by providing a proposition with context such as considering the conjunction of the proposition `(Database; isa(x, Table))` to `(Object; isa(x, Table))` from the example stated in previous section and having `(Database, Object; isa(x, Table))`. Formally we have
   $(\mathtt{c}_1; \mathtt{p}) \wedge (\mathtt{c}_2; \mathtt{p}) \Leftrightarrow (\mathtt{c}_1, \mathtt{c}_2; \mathtt{p})$
2. *Specializing:* by providing a context with propositions such as
   providing the proposition `(Object; isa(x, Furniture))` to the proposition `(Object; isa(x,Table))` and having `(Object; isa(x,Furniture), isa(x,Table))`. Formally we have
   $(\mathtt{c}; \mathtt{p}_1) \wedge (\mathtt{c}; \mathtt{p}_2) \Leftrightarrow (\mathtt{c}; \mathtt{p}_1, \mathtt{p}_2)$

Since we deal with propositions, the rules of first order and context logic apply. When the number of propositions is zero ($N = 0$ in Equation 1), then the vocabulary has its own context. For instance we have the list {`Office, Table, Book`}.

Another important possibility when rewriting the knowledge is that propositions are always asserted within other predicates in a recursive form. Henceforth given the general denotation of Equation 1, we have recursively $(\mathtt{c}_\mathtt{i}; (\mathtt{c}_\mathtt{j}; ., .), .)$ and subsequently `(Object; (Furniture; isa(x,Table)))`.

In general, the achievement in recursively rewriting the context and propositions deals directly with the critical and difficult step in context abstraction and is also a contribution of this paper. Although the problem of interoperating with recursive definitions is difficult to achieve with minimal inferencing, rewriting the context recursively has two advantages. (i) it maintains the connectivity of the knowledge and (ii) it provides one way to control the context abstraction. The latter is achieved by asserting one context for each predicate. The current implementation does not support recursive definition.

Another potential interest in recursive rewriting is that it converges to the Object Extended Model (OEM) formalism which has been widely used, namely as the interlingua for The Stanford-IBM Manager of Multiple Information Sources (TSIMMIS) [50]. OEM is a self describing object model with nested identity. Every object in OEM consist of an *identifier* and a *value*. The value is either atomic, or set of objects, denoted as set of {*label, id, value*}. We refer to the *label* and *value* as context and predicate respectively.

It should be noted that one of the innovations of the mediated architecture is that the proposed articulations need not be static. The partitioning of the domain knowledge is dynamic where articulation rules are asserted and retracted independently of the underlying knowledge base.

VI. Implementation: "Server-to-Server"

This section provides an overview of one implementation of the system capable of operating over the knowledge described by registered interfaces in CORBA interface repositories. The system is based on interfacing CORBA, HTML/XML type servers since XML is most likely to become the next industry standard for Internet client-server and distributed systems. The current generation of the system interoperates for Internet and HTML/XML (eXtensible Markup Language) servers, there is preference to present in this paper the complete original prototype which was written for CORBA applications only.

The "Mediator" software component that incorporates the domain algebra and articulations is a First Order Logic Production System. To allow a long-range vision and yet a fast development to market, off-the-shelf components were selected and hence the system was brought to meet numerous objectives. One of the objectives is to build an extendible prototype, a system that can supports non-traditional applications [5] and can serve as a environment for future innovations (e.g. when HTML/XML component was added on) and other improvements to enhance the information integration technology.

A. CORBA: A Brief Overview

The CORBA is a system of standards and specifications that describes how software components can interoperate across networks, languages and platforms [32]. CORBA allows for client-server interaction between heterogeneous objects distributed over a wide-area network. CORBA makes meta information describing the objects in a system and their interfaces available so that it can access other objects. Any object connected to an object Request Broker (ORB) can play simultaneously the role of client and server and hence Objects can initiate calls and respond to requests. ORB is the part of CORBA which facilitates client-server communication and interaction between distributed objects.

To reach object interoperability and for objects to plug and play, clients have to know exactly what they can expect from

[5] Currently Science Gate map genomic databases to diseases.

every object they might call upon for a service. In CORBA, the services that an object provides are described to interface between the object itself and the rest of the system. The objective of the interface is twofold: (i) it informs clients of the services that the objects provide as well as the access method to invoke these services and (ii) it informs the communications infrastructure of the format of the access methods.

B. The Mediator

The mediator was built using a productive development and delivery expert system tool which provides a complete environment for the construction of rules, knowledge and objects [40].

The mediator provides a cohesive knowledge representation tool for handling a variety of knowledge with support for three different programming paradigms: rule-based, object-oriented and procedural. The rule-based, are primarily intended for heuristic knowledge based on experience and hence articulation rules. On the other hand, the object-oriented programming allows complex systems to be modeled as modular components. The five generally accepted features of object oriented programming are class definitions, message handlers, abstraction, encapsulation, and inheritance. The object representation will be used to mirror hierarchically the declarative interfaces with object oriented flavor. The rules can interact and match objects. The procedural programming capabilities provide the necessary degree of freedom to expand for additional programming power. The system has Lisp like syntax to correspond to the First Order Logic.

The mediator includes a number of features to support the verification and validation of articulation rules, including support for modular design and partitioning of articulation rules, static and dynamic constraint checking of values and function arguments, and semantic analysis of rule patterns to determine if inconsistencies could prevent a rule interaction with the objects.

The mediator is based on *B-trees* indexes and a fast *Rete* pattern matching algorithm inherented from [40]. The efficiency of this algorithm is based on the assumption that data changes slowly over time. This assumption matches perfectly the nature of declarative interfaces that also slowly change over time.

C. The Dynamic Invocation Interface Functions

The Dynamic Invocation Interface (DII) allows requests to be built up and invoked dynamically to CORBA servers. Initially, clients need to know interface-related information only at the invocation time. A DII request, like a static request, is composed of an operation name, an object reference, and a parameter list.

In the current implementation, three functions were integrated in the system to support the DII requests. The objective is that given an object reference, the object's type and all information about that type can be determined at runtime by calling functions defined by the Interface Repository. In particular, these functions can determine: the module in which the interface was defined, if any, the name of the interface, the interface's attributes and their definitions, the interface's operations and their definitions, including parameter, context and exception definitions, and the inheritance specification of the interface.

GetCorbaInterface *server-name*

This function is to get all the interfaces from the Interface Repository of a CORBA server, whose name is the parameter of this function. It will get the necessary information through calls to the Interfaces Repository. Moreover, this function will then define all the interfaces locally. Although all the attributes and operations of the remote objects will also be put into the definition of the local objects, they are not used to store or retrieve values at this point. For instance, one may interrogate interfaces for their corresponding contents, given a `Factory` domain locates at IP address `171.64.75.95`,

```
(defcontext FACTORY)
(GetCorbaInterface ``171.64.75.95'')
```

and a `Shop` domain at an IP address `171.64.75.15`,

```
(defcontext SHOP)
(GetCorbaInterface ``171.64.75.15'')
```

Browsing the corresponding surrogates, we have

```
DOMAIN FACTORY

FACTORY
    FRAME
        Dimension
        Color-table
    SPOKE
    WHEEL
```

```
DOMAIN SHOP

SHOP
    BICYCLE
        Size
        Color
        stock-number()
    SUPPLIER
```

After *GetCorbaInterface* acquires the interfaces from the Interface Repository of the CORBA server and maps them locally, the system is ready to interoperate and create objects of the interfaces. More than one object of the same class can be created and each of them actually has its own context. The `defcontext` specification is equivalent to a module definition.

To keep track of all the references to CORBA objects created that were dynamically created, each local object references back the actual CORBA object to be able to retrieve attributes and invoke operations. This also maintains the independency and evolution of the servers. Hence a new attribute in every local object that corresponds to a remote CORBA object is created. This attribute is called `CorbaObjectNum`, which is a mapping between a local object and a remote object.

MakeCorbaInstance *interface-name object-name*

This function is needed for making instances of CORBA objects, and the new information is stored in the local objects. This information is called `CorbaObjectNum` and provides the mapping between the mediator objects and CORBA object references on the server. The full interface description will be put in a string, which is also stored as an attribute. This function takes two parameters: the class name and the object name. For instance one may interrogate the class `BICYCLE` in context `SHOP`, or more precisely `SHOP::BICYCLE` as

```
(MakeCorbaInstance SHOP::BICYCLE [Bontrager])
```

and hence a local surrogate object `[Bontrager]` is created. Each *MakeCorbaInstance* request updates the object references through `CorbaObjectNum`.

InvokeCorbaOperations *CorbaObjectNum operation return*

This function makes remote function calls to the objects residing on the CORBA server. It takes in the `CorbaObjectNum`, which provides the mapping between local mediator objects and remote CORBA object references on the server; the name of the operation that the user wants to invoke, and finally a multi-field value. This multi-field contains first the return type; then if the return is `OBJREF`, the return new object instance name

and class name; and finally a list of type/value pairs of parameters. Types can be `OBJREF`, `SHORT`, `LONG`, `USHORT`, `ULONG`, `FLOAT`, `DOUBLE`, `BOOLEAN`, `CHAR`, or `STRING`. *InvokeCorbaOperations* function calls the necessary CORBA functions to perform the operation dynamically. For instance, one may invoke a CORBA operation

```
(InvokeCorbaOperations
   (send [Bontrager] get-CorbaObjectNum)
   "get-stock-number"
   (FLOAT)).
```

The specific operation

```
(send [Bontrager] get-CorbaObjectNum)
```

gets the index into the CORBA object array to get the object reference. `"Get-stock-number"` is the desired operation to invoke on the object. In this example, `stock-number` computes the amount of items in stock. The multi-field `(FLOAT)` stores the information of the return type of the operation and information about the parameters, in this case a float value.

D. Articulation Rules: Bi-directional production Rules

A production rule facility allows definitions of operations that are executed whenever specific events occur or certain conditions are met. In general, a production rule takes the form of:

If [*condition*] **then** [*action*]

The approach taken by the production rules community has been to provide rules that take the activity in one direction, namely Left Hand Side (LHS) where conditions and patterns are described to the Right Hand Side (RHS) where the corresponding actions are listed [6]. On the other hand, articulations are equivalence rules and a new mechanism was adopted to reference them. In general, an articulation rule takes the form of:

[*condition-action*] **equivalent** [*condition-action*]

The syntax of the writing of articulation rules is based on an extension of production rule construction. An articulation rule is parsed in two directions and hence it becomes equivalent to two production rules. The current implementation of the articulation rule system includes three commands for defining and manipulating rules, namely *define-articulation*, *delete-articulation* and *modify-articulation*.

A production rule is activated when the condition is matched and the actions are executed. On the other hand, the actions modify the working memory according to the rule specifications. Since the mediator lookup is purely pattern-based, the triggering events are directly linked to the objects manipulated with the Dynamic Invocation Interface Functions, namely *GetCorbaInterface*, *MakeCorbaInstance* and *InvokeCorbaOperations*.

define-articulation *articulation-name partition* ⇔ *partition*

define-articulation is a mediator function that creates rules construct. This function effectively creates two production rules. For instance, a new rule, namely `new-rule`, can be defined as

```
(define-articulation new-rule
   (object (is-a FACTORY::FRAME)
      (Dimension ?x := (map ?x "10 01-04;.."))
   <=>
   (object (is-a SHOP::BICYCLE)
      (Size ?x := (map ?x "01-04 10;..")))
```

The `is-a` constraint is native to object-oriented relations and is used for specifying class constraints. This constraint also encompasses subclasses of the matching classes. On the other hand, the `:=` symbol is a new notation and is a predicate return value constraint operator. When needed, it is possible to use the return value of the external function `map` to modify the value of a field. In the conversion of an articulation rule to production rules, the predicate return value constraint operator is translated in the *action* list.

Redefining a currently existing articulation rule causes the previous rule with the same name to be removed.

delete-articulation *articulation-name*

The *delete-articulation* removes a previously defined articulation rule. Since an articulation is effectively two production rules, the deletion of an articulation rule is equivalent to the deletion of two production rules. The previously created rule `new-rule` can be deleted as

```
(delete-articulation new-rule)
```

modify-articulation *articulation-name partition* ⇔ *partition*

The *modify-articulation* action allows the partitions of articulation rules to be modified. The partitions of an articulation rules can be changed after the rule has been defined. However, this requirement is not enforced since the modify action is actually a deletion followed by the new definition. The *modify-articulation* is meant to complete the set of defining and manipulating the articulation rules.

```
(modify-articulation new-rule
   (object (is-a FACTORY::FRAME | DEPOT::FRAME)
      (Dimension ?x))
   <=>
   (object (is-a SHOP::BICYCLE)
      (Height ?x)))
```

The ``|'' constraint is a connective constraint. The mediator syntax allow three connective constraints, namely ``&'' (and), ``|'' (or) and ``~'' (not). The ``&'' constraint is satisfied if two adjoining constrains are satisfied. The ``|'' constraint is satisfied if either of the two adjoining constrained are satisfied. The ``~'' constraint is satisfied if the following constraint is not satisfied (Further details on predicate connective constraint and other features can be found in the the technical overview guide).

The two examples introduced in the *define-articulation* and *modify-articulation* cases demonstrate the mechanism of relating partitions. Since a partition may include more that one condition, conjunctive conditions can be listed with no constraint operator. The reason of listing two partitions with different contexts in the articulation rule as in the modify action examples demonstrate the multiple inheritance capability.

E. Operating over Domains

This section illustrates three examples of the domain ontology algebra relating to three operations, namely intersection, union and difference. The articulation rule examples introduced in Section VI-D can be alternatively edited in a user interface through tables. The intersection, union and difference examples in the current section refers to the set of articulation rules described in Table VI-E. Reading Figure VI-E from right to left, the articulation rules column displays the rule number. An interface name and class name are possible contexts for an entry. The predication occurs at the attribute level. Mapping functions are expanded to their contents.

1. *Domains intersection:* A `FACTORY` can query the domain `SHOP` and acquire the number of `BICYCLE` sorted by color and amount in stock.
2. *Domains union:* A consulting agency can verify for the domain `FACTORY` and acquire the number of `BICYCLE`, sorted by color and amount in stock, for more than one shop. This reflects a union over multiple domain intersections.
3. *Domain differences:* A `FACTORY` can query the domain `SHOP` and acquire the components or accessories that the `SHOP` relates to the `BICYCLE`, but `FACTORY` does not manufacture.

While intersection of multiple domain is comparably small to the domain themselves, placing the articulation rules in main memory is viable way to gain performance and thereby make it suitable for real time applications. These applications are envisages to interoperate with hundreds of domains.

VII. Status, Conclusions and Future Work

The current implementation is currently written in C/C++ ans integrates the C Language Integrated Production System [22][40]. Since user interface functions and data access functions are separated out into other components, the domain algebra consists mainly of rules.

The system as described in this paper is fully implemented and operational in Internet applications. The CORBA plugin component consists approximately of 4,000 lines of C/C++ code. The actual coding took about three man-months, but the system was carefully designed before any implementation began. The full blown mediator system exceeds the 150,000 lines of C/C++ code.

The fact that the system could be implemented in a short time reflects well on the integration aspects. This also underlines the intent of the author of system integration as opposed to coding. One intention of the implementation is that it can be used by researchers not involved in establishing domain algebras. The wrapper that translates CORBA interfaces to the mediator is currently made available as a public software package and can be down-loaded from (`www-db.stanford.edu/~maluf/-ccnp/ccnp.html`). The orginal CORBA wrapper was written jointly with Marcus Chan at Stanford University, To this stage the software has been down-loaded to over 100 researchers or users across the Internet.

This paper presents an approach that uses context formalism in the development of standard knowledge representations and knowledge sharing and plays a role in knowledge interoperability. The context approach provides a powerful tool to define the validity of knowledge relative to a situation. This paper address the problem of how to abstract and entail encoded knowledge within contexts.

We describe an environment to interface underlying knowledge resources to the outside world. The objectives set in this paper are to establish the intermediate model needed to sustain interoperability and to produce the needed environment. Hence, users and system developers can translate knowledge bases that provide comprehensive but simple coverage of topics of interest, knowledge usability and re-usability by different applications and knowledge maintenance in rapidly changing environments. The mediator can bring about a shift from merging knowledge to the manipulation, enhancement, and maintenance of domain intersections. The main objective of the mediator will be to handle an algebra that combines and partitions structures in a sound and well-behaved manner.

This paper describes a system of allowing multiple declarative interface interactions between heterogeneous objects distributed over a wide-area network. The objectives set in this paper are to establish the articulation needed for a domain algebra and thus sustain interoperation.

References

[1] S. Abiteboul and A. Bonner, "Objects and Views"; in Proceedings of the ACM SIGMOD International Conference on Management of Data"; Denver, Colorado, 1991.

[2] M. Aben, "KADS-II CommonKADS Inferences"; Technical Report M2-UVA-041-1.0, University of Amsterdam, 1993.

[3] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen and A. Verkamo, "Fast Discovery of Association Rules"; Advances in Knowledge Discovery and Data Mining, pp. 307-328, 1996.

[4] http://www.alexa.com, Navigate the web Smarter and Easier.

[5] A. Borgida, R. Brachman, D. McGuinness, L. Resnick, "CLASSIC: A Structural Data Model for Objects"; SIGMOD Conference pages 58-67, 1989.

[6] L. Brownston, R. Farrell, E. Kant and N. Martin, "Programming Expert Systems in OPS5: An introduction to Rule-based Programming"; Addison-Wesley, Reading Massachussetts, 1985.

[7] R. Brachman and J. Schmolze, "An Overview of the KL-ONE knowledge Representasion System"; Cognitive Science, 9(2):171-216, 1985.

[8] U. Fayyad, "Advances in Knowledge Discovery and Data Mining"; U Fayyad (Editor), ISBN: 0262560976, 1996.

[9] K. Bennett, U. Fayyad and D. Geiger, "Density-Based Indexing for Approximate Nearest-Neighbor Queries"; Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 1999.

[10] R. Cattell (editor), "The Object Database Standard: ODMG-93"; Morgan-Kaufmann Inc., Los Altos, California, 1994.

[11] S. Buvac and R. Fikes, "A Declarative Formalization of Knowledge Translation"; Knowledge Systems Laboratory, KSL-94-59, August 1994.

[12] S. Ceri and J. Widom, "Managing Semantic Heterogeneity with Production Rules and Persistent Queues"; in Proceedings of the Nineteenth International Conference on Very Large Data Bases, pages 108-119, 1993.

[13] N. Cercone, M. Morgenstern, A Sheth and W. Litwin, "Resolving Semantic Heterogeneity"; in Proceedings of the Sixth International Conference on Data Engineering, 1990.

[14] E. F. Codd, "A relational Model of Data for Large Shared Data Banks"; Communication of the ACM 13(6):377-387, 1970.

[15] C. Collet, M. Huhns and W. Shen, "Resource Integration Using a Large Knowledge Base in Carnot"; IEEE Computer, 12(24), Dec. 1991.

[16] A. Courtney, "Inter-Language Unification"; rel.1.5, Xerox PARC, 1994.

[17] N. Gehani and H. V. Jagadish, "Ode as an Active Database: Constraints and Triggers"; in Proceedings of the Seventeenth International Conference on Very Large Data Bases, pages 327-336, 1993.

[18] K. Chang, H. Garca-Molina and A. Paepcke, "Boolean Query Mapping Across Heterogeneous Information Sources"; IEEE Transactions on Knowledge and Data Engineering, 8(4):515-521, Aug, 1996.

[19] R. El-Masri and G. Wiederhold: "Data Model Integration Using the Structural Model"; Proceedings 1979 ACM SIGMOD Conference, pages 191-202.

[20] M. Genesereth and N. Nilsson, "Logical Foundations of Artificial Intelligence"; Morgan Kaufmann Publishers, Inc., Los Altos, California, 1987.

[21] M. Genesereth, N. Singh and M. Syed, "A Distributed and Anonymous Knowledge Sharing Approach to Software Interoperation"; Third International Conference on Information and Knowledge Management, 1994.

[22] J. Giarratano and G. Riley, "Expert Systems: Principles and Programming"; Second Edition, Boston, PWS Publishing Company, 1994.

[23] T. R. Gruber, "Ontolingua: A mechanism to support portable ontologies"; Technical Report, KSL-91-66, Knowledge System Laboratory, Stanford University, Stanford, CA.

[24] R. V. Guha. "Context: A Formalization and Some applications"; Doctoral Dissertation, Stanford University, 1991.

[25] W. Litwin, L. Mark and N. Roussopoulos, "Interoperability of Multiple Autonomous Databases"; ACM Computing Surveys, 22(3):267-293, 1990.

[26] M. Desmarais, D. A. Maluf and J. Liu, "User-Expertise Modeling with Empirically Derived Probabilistic Implication Networks,"; User Modeling and User-Adapted Interaction Journal, Kluwer Academic Publishers, Vol. 5, No. 3-4, pp. 283-315, 1996.

[27] D. A. Maluf, " Implementing Articulation Rules for Object Request Brokers as an Extension to Production Systems"; IEEE Knowledge and Data Engineering Exchange Workshop, Newport Beach, California, pp. 36-44, 1997.

[28] D. A. Maluf and G. Wiederhold, "Abstraction of Representation for Interoperation"; Tenth International Symposium on Methodologies for Intelligent Systems, Lecture Notes in Computer Science, Springer Verlag, pp 441-455, October, 1997.

[29] `www-db.stanford.edu/~maluf/ccnp/ccnp.html`

[30] D. Lenat and R. Guha, "The Evolution of CycL, The Cyc Representation language"; Special Issue on Implemented Knowledge Representation System, Sigart, ACM, 2(3), pages 84-87, June 1991.

[31] J. McCarthy, "Notes on Formalizing Context"; In proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, 1993.

[32] J. Mowbray and R. Zahavi, "The Essential CORBA"; John Wiley and Sons, 1995.

Predicate		Context		Articulation Rule
Translation Function	Attribute	Class	Interface	
01-04 yrs = 10" 05-08 yrs = 14" 09-11 yrs = 16"	Size	BICYCLE	SHOP	0123
12-15 yrs = 18" 16-18 yrs = 20" 18-65 yrs = 22"	Dimension	FRAME	FACTORY	
#FC2334 = Black #FC3365 = Red #FC5467 = Blue	Color	BICYCLE	SHOP	0125
#FC8934 = White #FC2422 = Yellow #FC2545 = Grey	Color-table	FRAME	FACTORY	

Fig. 1. Articulation rules can be alternatively edited in a user interface through tables. From right to left, the articulation rules column displays the rule number. An interface name and class name are possible contexts for an entry. The predication occurs at the attribute level. Mapping functions are expanded to their contents. Hence, a mapping function that translates units of `age` to `inches` can be stated as facts.

[33] http://www.netnanny.com, Policing the Cyberspace.

[34] National Information Standards Organization, "Information Retrieval (Z39.50): Application Service Definition and Protocol Specification (ANSI/NISO Z39.50-1995)"; NISO Press, Bethesda, MD, 1995. Accessible at `http://lcweb.loc.gov/z3950/agency/`.

[35] R. MacGregor and R. Bates, "The LOOM Knowledge Representation Language"; in Proceedings of the Knowledge-Based Systems Workshop, April 1987.

[36] N. Singh, and M. Gisi, "Coordinating Distributed Objects with Declarative Interfaces"; in Coordination Languages and Models, Lecture Notes in Computer Science, 1061, Springer, 1996, pages 368-385.

[37] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patill, T. Senator, and W. R. Swartout, "Enabling technology for knowledge sharing"; AI Magazine, 12(3):36-55, 1991.

[38] Object Management Group, "The Common Object Request Broker: Architecture and specification"; Accessible at `ftp://omg.org/pub/CORBA`, December 1993.

[39] R. Orfali, D. Harkey and J. Edwards, "The Essential Distributed Objects Survival Guide"; John Wiley and Sons, 1996.

[40] G. Riley, "CLIPS: An Expert System Building Tool"; Proceedings of the Technology 2001 Conference, San Jose, CA, December 1991.

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

[42] Science gate Bay `http://sciencegate.com`

[43] T. Sellis, C. Lin and L. Rashid, "Implementing Large Production Systems in a DBMS Environment: Concepts and Algorithms"; in Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, 1991.

[44] G. Wiederhold, T. Barsalou, B. S. Lee, N. Siambela and W. Sujansky, "Use of Relational Storage and a Semantic Model to Generate Objects: The PENGUIN Project"; Database '91: Merging Policy, Standards and Technology, The Armed Forces Communications and Electronics Association, Fairfax VA, June 1991, pages 503-515.

[45] G. Wiederhold and M. Genesereth, "The Conceptual Basis for Mediation Services"; to appear in IEEE Expert, 1997.

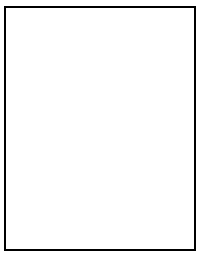
[46] G. Wiederhold, "An Algebra for Ontology Composition"; Proceedings of 1994 Monterey Workshop on Formal Methods, U.S. Naval Postgraduate School, Monterey CA, pages 56-61, 1994.

[47] G. Wiederhold, "Interoperation, Mediation, and Ontologies"; International Symposium on Fifth Generation Computer Systems (FGCSOB94), Workshop on Heterogeneous Cooperative Knowledge-Bases, Vol 3, pages 33-48, ICOT, Tokyo, Japan, 1994.

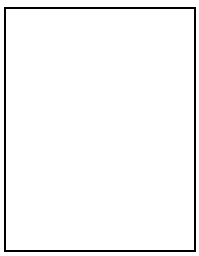
[48] G. Wiederhold and X. Qian, "Modeling Asynchrony in Distributed Databases"; Third IEEE Computer Society Data Engineering Conference, Los Angeles, Feb. 1987.

[49] Extensible Markup Language (XML). Accessible at `http://www.w3.org/-TR`, W3C Technical Reports and Publications.

[50] Y. Papakonstantinou, H. Garcia-Molina and J. Widom, "Object Exchange Across Heterogeneous Information Sources"; International Conference on Data Engineering, 1995.



David A. Maluf received the Master of Engineering and Ph.D. degrees in Electrical Engineering from McGill University in 1991 and 1995, respectively, as well as his premedical studies from the faculty of Sciences at McGill University. He worked for several years as an advisor, research associate, and senior researcher at both Center de Recherche Informatique de Montreal and the government in Canada, prior to joining Stanford University in 1996, where he was involved in the database group. He was adjunct professor in the faculty of engineering at McGill University in Canada. He has been involved in academic and industrial research relating to the area of: automation, intelligent information integration, databases, knowledge discovery, image understanding, and have yielded in several millions of research dollars. His focus on data intensive science problems in bio-informatics resulted in the first solution allowing high throughput integration of genomic data for accelerated analysis. His technique has been integrated with Incyte Pharmaceuticals propriety products and Science Gate Bay of which he founded and is currently on the board of director. Currently he is with Computational Science Division at NASA Ames Research and the Chief Information Officer for NASA's Engineering for Complex System Program.



Peter B. Tran is a graduate of the University of California at Davis in Electrical Engineering. He worked as a consultant, architect, and software engineer at several companies, such as BEA Systems, XUMA, Computer Sciences Corp., and Recom Technologies, emphasizing on distributed, web-based applications and information management. He is currently a member of the Aerospace ExtraNet (AEN) group of the Computational Sciences Division at NASA Ames Research Center where he is working on Intelligent Information Integration and Management projects.