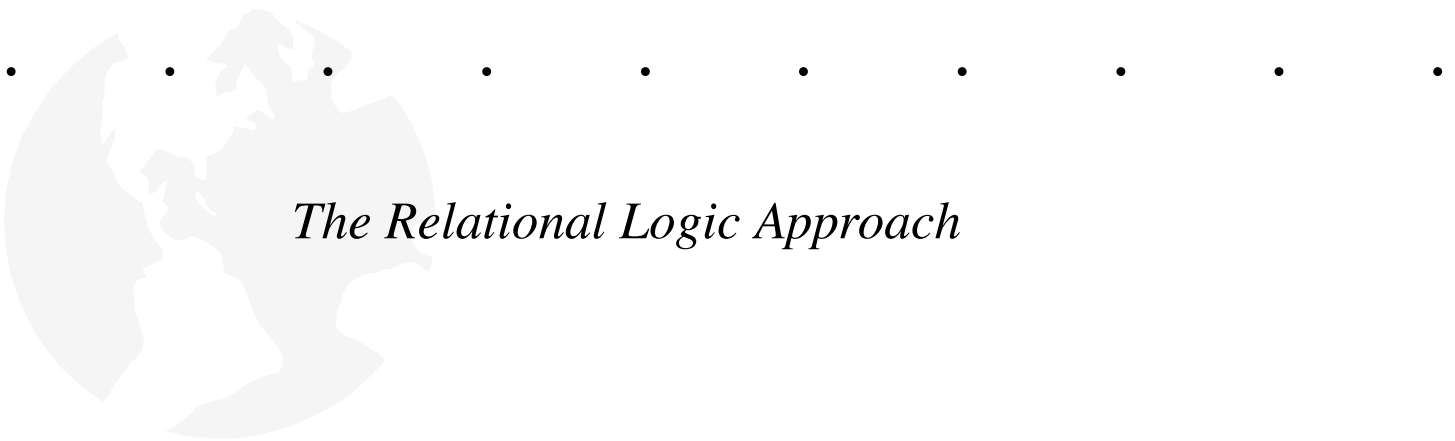


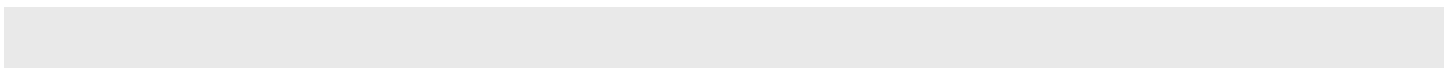
⋮
mergent systems

Semantic Integration of Heterogeneous Databases



The Relational Logic Approach

Michael R. Genesereth
genesereth@mergent.com



⋮

Semantic Integration of Heterogeneous Databases

The Relational Logic Approach

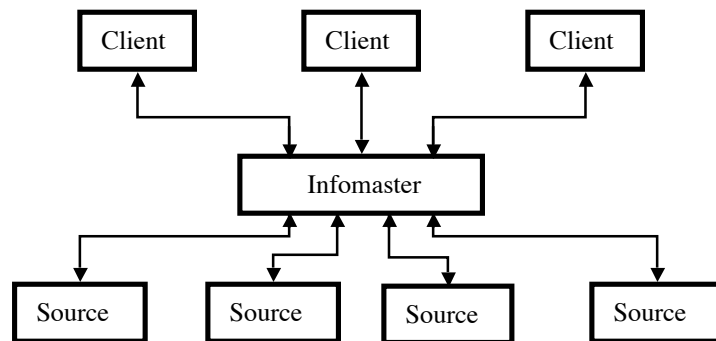
Introduction

In recent years, there has been a dramatic growth in the number of databases available on computer networks. The growth rate is even greater if we expand our definition of a database to include other structured information sources, such as XML files and application programs.

Unfortunately, utilizing these data sources is not always easy. Frequently, the information necessary for a particular task is distributed over multiple databases; and, in such situations, users must expend the effort to find those databases, extract the relevant fragments of data, and aggregate the results into a coherent whole.

Infomaster is a database integration system that solves this problem. It provides integrated access to multiple, distributed databases, giving each user the illusion of a single, centralized database specific to his or her need.

The diagram shown below illustrates the architecture of an integrated information system using Infomaster. In the example shown here, there are just three clients and four sources.



As suggested by the diagram, the client communicates with Infomaster, and Infomaster takes the responsibility of finding appropriate sources, extracting the relevant information, and returning the overall result to the client.

Note that the client in such a system can be a human user interacting through a web browser; it can be an application program treating the broker as a virtual database; it can be a data warehouse using the broker to update its information. Similarly, the sources can be of various types. They can be relational databases (accessible, for example, via ODBC), files (tab-delimited text, XML, and so forth), or application programs (using LDAP, ADAP, etc.).

Conceptual Heterogeneity

One of the key problems in building an integrated information system stems from *conceptual heterogeneity*, i.e. differences in the vocabulary and structure of different data sources.

Examples of conceptual heterogeneity abound, from simple cases of incompatible units to complex cases involving major structural differences.

As an example of medium complexity, consider the two genealogical databases shown below. In the first database, there is a single relation, expressed here as a table in which each row represents a person and in which there are different fields for the person's father and mother.

person	father	mother
bob	art	amy
cal	bob	bea
coe	bob	bea

In the second database, there are two relations, one with parent information and the other with gender information.

parent	child
art	bob
amy	bob
bob	cal
bea	cal
bob	coe
bea	coe

person	gender
art	male
amy	female
bob	male
bea	female

The information in these two databases is the same, but the way in which it is represented is different. Such differences make it difficult for an information broker to locate databases, extract information, and aggregate that information.

Some have argued that the problem of conceptual heterogeneity can be eliminated by the development of standard data schemas. Unfortunately, this is not the case. One reason is that the standards process is time-consuming; and, as a result, standards usually lag actual usage, coming only after years of work and changing only very slowly in response to new developments. Another problem is that standards usually apply only to specific application areas. In our modern information infrastructure, information processing must often cross application area boundaries, for which joint standards are not likely to exist. So, to solve the problem of conceptual heterogeneity, we must look elsewhere.

Semantic Integration

Semantic integration is the process of harmonizing conceptual differences between disparate databases. This can be done in two ways -- through data transformation or through query transformation. In the data transformation approach, data is transformed from a set of source schemas into a target schema prior to being used to answer queries. In the query transformation approach, queries in a target schema are transformed into subqueries in appropriate source schemas. The choice between these alternatives is affected several factors, including

updateability and query performance. In either case, the conceptual differences between source and target schemas must be eliminated.

In this section, we compare and contrast three approaches to semantic integration, viz. procedural integration, relational algebra, and relational logic. In discussing these approaches, we consider the query transformation alternative only; the issues are much the same for the data transformation alternative.

In the *procedural* approach to semantic integration, the idea is to write software to compute target relations from source relations. Any high-level programming language can be used.

The advantage of this approach is that solving problems through programming is a well-understood methodology. For many programming languages, there are well-developed programming techniques and tools, and there are many people trained to write programs.

The main disadvantage of programming is that it is more costly and time-consuming than necessary. Schema transformation does not require the full power of a general-purpose programming language nor the skills of a general-purpose programmer. It is better done by database administrators with knowledge of the schemas involved, using schema transformation tools rather than general-purpose programming tools.

In the *relational algebra* approach to semantic integration, target relations are defined as "views" of source relations in a language based on relational algebra, the most popular of these languages being SQL.

On the face of it, this approach has significant merit. Being more declarative and more attuned to the problem of schema mapping, SQL simplifies the task of semantic integration and brings it within the skill set of database administrators. Moreover, there is an economy of scale. Database system vendors can implement query optimization techniques once, saving integrators the effort of programming these techniques into every piece of transformational software.

Unfortunately, the relational algebra approach has its own problems. The most serious of these stems from the limited expressiveness of relational algebra languages (like SQL). There are common cases of target schemas that simply cannot be defined in terms of source schemas using SQL.

As an example of this, consider the case of two relations providing information about personnel. Each has three attributes. The first relates social security number, department, and phone; the second relates social security number, department, and salary. Neither of these relations can be defined in terms of the other. The first cannot be defined in terms of the second because the second does not contain phone information. The second cannot be defined in terms of the first because the first does not contain salary information. However, both contain department information; and, for the purposes of integration, it would be desirable to capture this relationship.

The *relational logic* approach to semantic integration is similar to the relational algebra approach in its use of a declarative language designed specially for schema mapping. However, the language of relational logic is more expressive than SQL; and, as a result, it can capture relationships like the one in the preceding example.

The next few sections describe the relational logic approach in greater detail, starting with the notion of relational rules, proceeding through a discussion of rule application and rule inversion, and finally showing how rules and inversion work together in model-centric semantic integration.

Relational Logic

The heart of relational logic is an expressive language for writing "rules" that characterize output relations in terms of input relations. As we shall see in the next section, there is an algorithm for "applying" rules written in this language. Moreover, as we shall see in the section after that, there is a technique for "inverting" rules to produce new rules that characterize input relations in terms of output relations.

The syntax and semantics of rules in relational logic are most easily introduced in stages. In the following paragraphs, we look at "basic" rules first, then "existential" rules, and finally "disjunctive" rules.

A *basic rule* is an expression of the form shown below. Here, p and p_1, \dots, p_k are names for relations, and the various t_{ij} are variables or constants. $p(t_1, \dots, t_n)$ is called the *conclusion* of the rule, and the expressions $p_1(t_{11}, \dots, t_{1n}), \dots, p_k(t_{k1}, \dots, t_{kn})$ are called *conditions*.

$$p(t_1, \dots, t_n) \text{ :- } p_1(t_{11}, \dots, t_{1n}), \dots, p_k(t_{k1}, \dots, t_{kn})$$

Semantically, a rule is something like reverse implication. It is a statement that the conclusion of the rule is true whenever the conditions are true (once constants have been substituted in a consistent way for all variables).

As an example of a basic rule, consider the expression shown below. This rule defines the grandparent relation in terms of the parent relation. A person X is the grandparent of a person Z if X is the parent of a person Y and Y is the parent of Z .

$$\text{grandparent}(X, Z) \text{ :- } \text{parent}(X, Y), \text{parent}(Y, Z).$$

In the standard notation for relational logic, variables begin with upper case letters, and constants begin with lower case letters or digits; so all of the arguments here are variables.

Note that the same relation can appear as a conclusion in more than one rule. For example, parenthood can be defined in terms of fatherhood and motherhood as shown below.

$$\text{parent}(X, Y) \text{ :- } \text{father}(X, Y).$$

$$\text{parent}(X, Y) \text{ :- } \text{mother}(X, Y).$$

All of relational algebra can be captured by rules of this form (with extensions to express conditions on the absence of information). Going beyond relational algebra requires existential rules and/or disjunctive rules.

The need for *existential rules* arises when it is necessary to assert the existence of an object without knowing its identity. In relational logic, we can refer to such an anonymous object using a "Skolem" term, viz. an arbitrary function constant applied to the variables in the rule.

As examples of this, consider the two expressions shown below. Together, they characterize the parent relation in terms of the grandparent relation, i.e. the inverse of the definition given above. If X is the grandparent of Z , then there must be some person such that X is the parent of that person and such that that person is the parent of Z .

$$\text{parent}(X, f(X, Z)) \text{ :- } \text{grandparent}(X, Z).$$

$$\text{parent}(f(X, Z), Z) \text{ :- } \text{grandparent}(X, Z).$$

Skolem terms are similar to "null values" as used in relational databases. The main difference is that Skolem terms include variables to make explicit the dependence between each anonymous object and the objects on which it depends. As a result, we can safely equate syntactically identical Skolem terms (because they must refer to the same objects) whereas it is never safe to equate null values (since they may refer to different objects).

The need for *disjunctive rules* arises when there is more than one possible conclusion for a given set of conditions and it is not clear which conclusion holds. In relational logic, this situation is handled by writing multiple conclusions on the conclusion side of the rule.

The expression shown below is an example of a disjunctive rule. It states that, if X is a parent of Y, then either X is the father of Y or X is the mother of Y.

```
father(X,Y) | mother(X,Y) :- parent(X,Y)
```

In this case, only one of the conclusions can follow, since a person cannot be both a father and a mother. In general, it is possible that both of the conclusions hold.

Skolem terms and disjunction provide the necessary expressiveness to extend relational algebra in the ways necessary to support semantic integration. In the section on rule inversion, we see how these features are introduced when some definitions are inverted.

Rule Application

There are multiple ways of computing with rules. In what follows, we look at a "bottom-up" method of computation. The method is so-named because we start at the "bottom" with given tables and work our way "up" to derived tables. There are more efficient methods of computation, but this one is simpler to understand than others.

The computation starts with a set of relational tables. The tables are rewritten as sentences, one sentence for each row of each table. This is called the active set. The figure below shows the sentences corresponding to tables for the father relation and the mother relation.

```
father(art,bob)
father(bob,cal)
father(bob,coe)
mother(amy,bob)
mother(beatrice,cal)
mother(beatrice,coe)
```

The computation goes as follows. The conditions of each rule are compared to the sentences in the active set to find variable bindings such that all instantiated conditions are in the active set. If this can be done, then the instantiated conclusion of the rule is added to the active set. (Instantiated conditions and conclusions are obtained by replacing all variables by their bindings.) This process continues until there are no new conclusions to add to the active set.

As an example, suppose that we had the following rules.

```
parent(X,Y) :- father(X,Y)
parent(X,Y) :- mother(X,Y)
grandparent(X,Z) :- parent(X,Y), parent(Y,Z).
```

Starting with the father and mother tables shown above, we can compute the parent tables by "matching" the conditions in the first two rules to the rows of these tables to produce the following additional conclusions.

```
parent(art,bob)
parent(bob,cal)
parent(bob,coe)
parent(amy,bob)
parent(bea,cal)
parent(bea,coe)
```

Continuing this process, we can compute the grandparent relation by matching the conditions in the third rule to this derived table in all possible ways.

```
grandparent(art,cal)
grandparent(art,coe)
grandparent(amy,cal)
grandparent(amy,coe)
```

The computational method is no different in the presence existential and disjunctive rules. However, the answers are different in that they can contain Skolem terms or disjunctive conclusions.

Note that such complex expressions are necessary during rule application and may lead to simple answers (i.e. those sentences not containing Skolem terms or disjunction). However, they are usually undesirable in replies to users. Thus, standard practice in such situations is to compute answers by considering only the simple sentences in the resulting active set.

Note also that, because of Skolem terms, this process may not terminate in the presence of recursion. Recursion means that there are relations that are defined in terms of themselves, either directly or indirectly. In such situations, the system may need to cut off the computation arbitrarily and may thereby lose answers. For relational rules in general, this is a problem. Fortunately, for model-centric integration as described below, even though there may be recursion, this computation always terminates after a finite amount of time.

Finally, in thinking about rule application, it is important to bear in mind that, as mentioned above, this method, while being simple to understand, is not especially efficient. Much more efficient computation methods are known and are typically used in rule application systems.

Rule Inversion

One of the key features of Infomaster is its ability to invert rules. If one writes rules in one direction, Infomaster can automatically write corresponding rules in the opposite direction. The rules resulting from this inversion process are guaranteed to produce only correct answers, and they are guaranteed to produce all correct answers. This is remarkable in that the inverses of some definitions cannot be expressed in languages like SQL and the closest approximation in SQL produces incorrect answers.

As an example of this capability, consider the parenthood example shown below. A person X is the father of a person Y if X is a parent of Y and X is male. A person X is the mother of a person Y if X is a parent of Y and X is female.

```
father(X,Y) :- parent(X,Y), gender(X,male).
```

```
mother(X,Y) :- parent(X,Y), gender(X,female).
```

If we are given a parent table and a gender table, we can use these rules to compute tables for the father relation and the mother relation, say to store them in our data warehouse. Now, suppose our network goes down and the parent and gender tables become unavailable. No problem with father and mother since we have them stored locally. But what if someone asks for parent information or gender information?

The good news is that the rules above can be inverted to give definitions for parent and gender in terms of father and mother. The rules below show the results of the inversion process in this case. X is a parent of Y if X is the father of Y or if X is the mother of Y. We know that X is male if we know that X is known is someone's father, and we know that X is female if we know that X is someone's mother. Thus, we have a way of computing parent and gender from father and mother.

```
parent(X,Y) :- father(X,Y).
```

```
parent(X,Y) :- mother(X,Y).
```

```
gender(X,male) :- father(X,Y).
```

```
gender(X,female) :- mother(X,Y).
```

In this case, complete inversion is possible using only basic rules. Unfortunately, this is not always the case.

If the conditions in the rules defining a relation mention variables that are not used in the conclusion, then the inversion will contain a Skolem term. The inversion of the grandparent relation defined earlier is a good example. Inversion leads to the two rules shown below.

```
parent(X,f(X,Z)) :- grandparent(X,Z).
```

```
parent(f(X,Z),Z) :- grandparent(X,Z).
```

If there is more than one rule defining a relation, then the inversion will contain a disjunction. The definition of the parent relation in terms of father and mother is a good example. In the absence of gender information, the best we can do is the following disjunctive rule.

```
father(X,Y) | mother(X,Y) :- parent(X,Y)
```

In such cases, information is lost when one moves from a source schema to a target schema, because the target schema is unable to represent all of the information in the source schema. The good news is that, at least for basic rules, the inversion process produces relational rules that provide the maximal amount of information possible. No one, using any approach to integration, can do better.

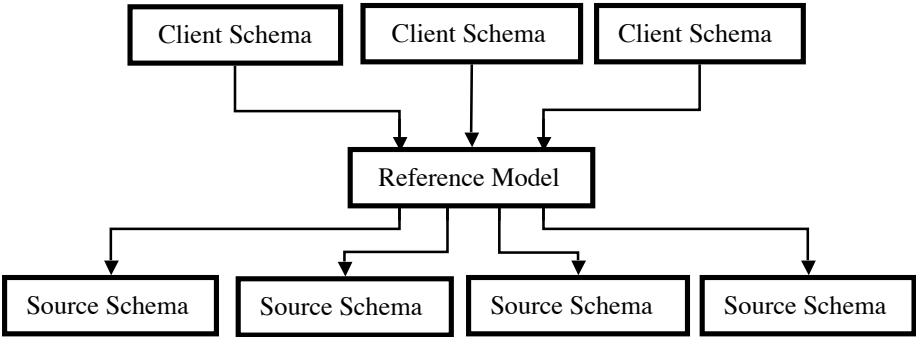
While inversion has benefit in any integration situation, the real benefit is most apparent when used in conjunction with model-centric integration, as discussed in the next section.

Model-Centric Integration

As described thus far, semantic integration is done on a pair-wise basis. This works fine when there are just two schemas to be integrated. However, what happens when there are ten schemas?

Is it necessary to write nine programs or views or rules for each schema, making a total of ninety pieces of work? What happens when there are a thousand schemas or more? Because of the multiplicative character of pair-wise integration, it is often said to suffer an "n-squared" problem.

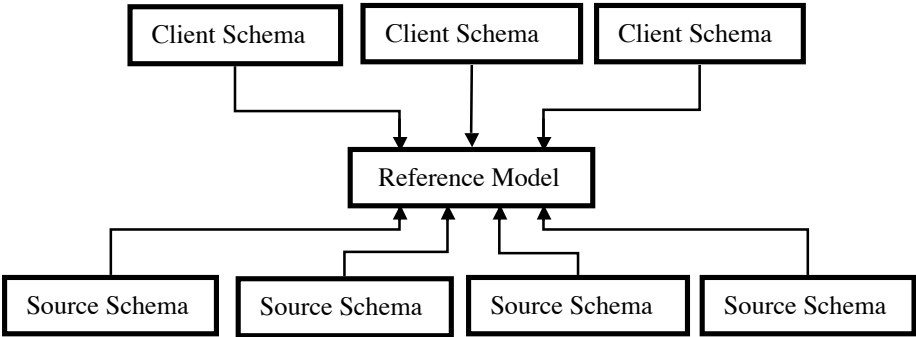
The answer to this problem is the use of a reference schema for an application area. In this approach, each client and each source schema is related to a common reference schema, as shown below.



Since there is only one conversion for each client and each source, the cost of this approach is linear in the number of schemas to be integrated -- ten schemas, ten conversions.

As it turns out, a variation on this idea can also be used to solve another problem, viz. ease of use. The strength of relational logic, with respect to SQL, is its superior expressiveness. Unfortunately, this expressiveness is not without its drawbacks. Some users find the additional features of the language difficult to use.

The good news is that it is possible to have the benefits of relational logic without this extra complexity. The answer is to provide users with a reference schema and allow database administrators to write basic rules defining their relations in terms of the relations in this reference model. The following diagram illustrates this idea.



A well designed reference is more detailed than any of the schemas it is used to integrate. The importance of this is that it is possible for administrators to define client and source relations in terms of reference relations using just basic rules in relational logic (or even SQL). They need not write existential or disjunctive rules at all!

Of course, in order for a system to answer queries, the rules defining source relations in terms of reference relations must be reversed. This is where inversion comes in. Infomaster can use its view inversion capabilities to invert these rules, thus providing rules for the client relations directly in terms of the source relations. Of course, the output of this procedure is likely to include existential and disjunctive rules, which is why relational logic is so important.

As an example of how this works, consider the example given earlier to illustrate the expressive restrictions of SQL. Neither of the two relations can be defined in terms of each other using SQL (though they can be interrelated in relational logic). However, they can both be defined in terms of a combined relation in a detailed reference model. If we name the two relations p and q and add a relation r to our reference model that combines these two relations, we can write the following definitions. Here, both p and q are defined in terms of r . There are no rules relating p to q or vice versa.

$$p(SSN, DEPT, PHONE) :- r(SSN, DEPT, PHONE, SALARY)$$

$$q(SSN, DEPT, SALARY) :- r(SSN, DEPT, PHONE, SALARY)$$

Now, suppose that relation q is stored explicitly but p is not and we would like to answer questions about p . By inverting the rule relating r and q , we get the following relational rules. Note the Skolem term in the second rule.

$$p(SSN, DEPT, PHONE) :- r(SSN, DEPT, PHONE, SALARY)$$

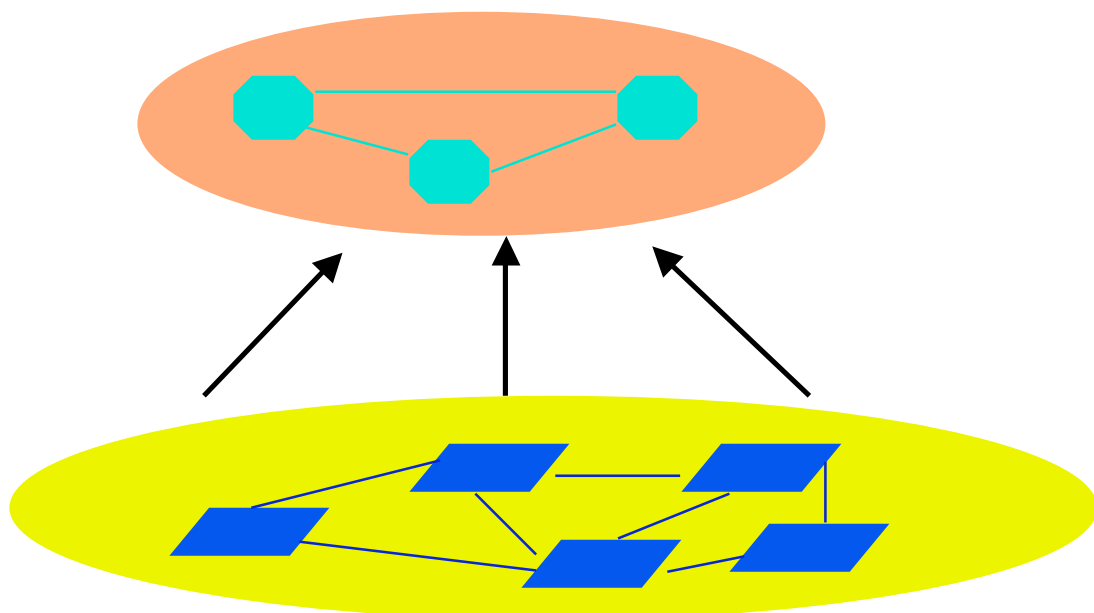
$$r(SSN, DEPT, f(SSN), SALARY) :- q(SSN, DEPT, SALARY)$$

If we are asked for the department of a specific individual (as a projection of the p relation), we can use these relational rules to get an answer. Of course, if we are asked for a person's phone, no answer is possible since q does not store this information and, by our assumption, neither p nor r are explicitly stored.

Corporate Datawebs™

One of the most exciting by-products of the model-centric approach to semantic integration is the notion of a dataweb, a highly effective user interface for heterogeneous data systems. A dataweb has two defining characteristics.

First of all, a dataweb is a virtual information network built on top of the physical information sources available on a network. The information in each node of a dataweb can be drawn from multiple physical sources.



The second and, in many ways, more important feature of a dataweb is its object-oriented character. Each node represents a single entity in an enterprise, e.g. a person, a location, an organization, a computer, etc. The conceptual structure of the dataweb is given by the enterprise model used in data integration. If the enterprise model is well designed, it is possible to use semantic integration to provide a user interface with the following features.

Clicking on an identifier brings up a page containing information about the associated entity. Each such page contains information about that one entity only and contains all of the relevant information about that entity. Clicking on an identifier within an entity's page brings up a page of information about the associated entity.

Coordinated with this browsing capability is a powerful semantic search tool. Using this tool one can search for entities by specifying properties of the entity, properties of related entities, properties of those properties, and so forth. What's more, these complex queries can be created by end users with minimal training.

Datawebs can support update. A user can supply authorized updates by changing information directly on the pages presented to by the system. This is sometimes called WYSIWYC (What You See Is What You Change), and this embodies the so-called Single Entry Principle (you change your phone number in just one place, and the system automatically distributes that change to the relevant databases).

Finally, there is a coordinated notification capability. If a user can ask a query, he can also ask to be notified about future states of affairs by clicking the Notify button instead of the Display button.