MCC Technical Report Number ACT-OODS-073-91-Q

# Semantic Integrity and Integration Support in Carnot

Christine Collet and Michael N. Huhns

March 6, 1991

**MCC/ACT Confidential and Proprietary**

**Abstract**

The goal of the Semantic Integrity and Integration Support Project is to develop methods for integrating separately developed information resources to enable them to be accessed and modified coherently. We have chosen the composite approach to integration, in which transactions are written against a global schema before they are translated and distributed to each information resource. However, we use the Cyc knowledge base for this global schema, rather than construct a new one for each integration task. This paper presents an analysis of the resource integration problem, including previous approaches to a solution and their limitations. We focus on the properties used to represent the semantics of a resource, which are the key to its integration. We then describe our proposed approach and a plan for its development and evaluation.

Microelectronics and Computer Technology Corporation
Advanced Computing Technology Program
3500 West Balcones Center Drive
Austin, TX 78759-6509
(512) 338-3651
huhns@MCC.COM

# Contents

# 1 Objective

The goal of the research described in this proposal is to develop methods for integrating separately developed information resources to enable them to be accessed and modified coherently. The methods will provide logical connectivity among the information resources via a semantic communication layer that automates the maintenance of data integrity, and provides an approximation of global data integration across systems. The research constitutes a fundamental and integral part of the Carnot Project, which is addressing the problem of logically unifying the information in a physically-distributed, enterprise-wide information space. If successful, the Carnot Project will provide a user with the capability to navigate through information efficiently and transparently, update the information consistently, and write applications easily for such an information space.

The need for this capability is critical. Strategic business applications that require intercorporate linkage (e.g., linking buyers with suppliers) or intracorporate integration (e.g., producing composite information from engineering and manufacturing views of a product) are becoming increasingly prevalent. Unfortunately, the proliferation of personal workstations, departmental servers, and geographically distributed mainframe computers has led to an unavoidable *decentralization* of information; corporate computing environments have in the process become heterogeneous, with information spread across dissimilar platforms, applications, and media. This decentralization has resulted in consistency problems among the information resources. Current attempts to deal with this inconsistency have been expensive, error prone, and ad hoc.

In this proposal, we analyze the requirements of a system for achieving the semantic layer and describe a tool for supporting its development in an organization. The tool will incorporate facilities to specify and maintain the semantics of an organization's integrated information resources. The system and tool resolve semantic inconsistency among disparate information resources by using an existing global ontology: the Cyc knowledge base [Lenat and Guha 1989]. Cyc encodes the semantics for a significant portion of human consensus reality to which the semantics for each information resource can be related. Cyc also provides the representation and inference mechanisms needed for expressing the relationships among the information resources. The remainder of this section discusses the semantic integrity and

integration problem, describes prior attempted solutions, and introduces our initial approach to solving it for Carnot.

## 1.1 Semantic Integrity and Integration

Today's large organizations have many independent information resources. Because the information resources must serve the needs of various applications, there are many different types, such as a database management system with its database(s), an information repository, an expert system with its knowledge base, or an application program with its data and productions (such as an application generator, a report generator, or a spread sheet) [Navathe *et al.* 1989]. These different types of resources are largely incompatible with each other in syntax and in formal semantics. Additional incompatibilities and heterogeneities arise due to

- different hardware and operating system software.
- different physical and logical data structures. Different formats are used to represent information at the physical level (block or page), conceptual level (such as object-oriented, relational, network, file, or hierarchical model), and external level. Differences at the external level are strictly related to the data structures of the databases and the programming languages provided by the database management system.
- different organizations having (informal) semantics about the real world that differ due to culture, management rules, language spoken, etc. Information resources all attempt to model some portion of the real world, and necessarily introduce simplifications and inaccuracies in this attempt that result in incompatibilities.

By integrating heterogeneous resources in a single environment, an application or a user may interact with this environment to request and update information and, more generally, execute tasks dependent on different resources. But creating such an environment requires that the incompatibilities be resolved. Several methods have been devised for resolving incompatibilities that arise during query, update, and maintenance operations. We describe some of these methods next.

### 1.1.1 Query Operations

Two approaches have been suggested to solve the problem of providing integrated access to a collection of existing, heterogeneous databases connected over a network [Buneman *et al.* 1990]. The first approach, called the *composite approach*, introduces a monolithic global/virtual schema that describes the information in the databases being composed. Database access and manipulation operations are expressed in a universal language and are then mediated through the new conceptual schema. Through this schema, the users and applications are presented with the illusion of a single, integrated, centralized database. They need not be aware of semantic conflicts of facts that may exist among the databases, because explicit resolutions for the conflicts can be specified in advance.

However, a global schema is difficult to construct. Moreover, if any of the underlying local database schemas change or if any new local schema is added, the integration process must be performed again. This approach also cedes control over the structure of existing databases to a central authority. Further, the interface can be used only if all databases needed to perform a task are accessible.

The second approach, called the *federated approach* [Buneman *et al.* 1990] or the *autonomous approach* [Ahlsen and Johannesson 1990] and [Litwin *et al.* 1990], avoids constructing a global schema, and merely presents the user with a collection of local schemas, along with tools for information sharing among databases. The user resolves conflicts of facts in a manner particular to each application, and integrates only the portions of the databases that are necessary. [Ahlsen and Johannesson 1990] also define three types of autonomy that can be exploited: 1) network autonomy, in which there is no central authority for communications, 2) behavioral autonomy, in which there is local control of processing, and 3) semantic autonomy, in which there is no global schema. The advantages cited for this approach include increased security, easier maintenance, and the ability to deal with inconsistent databases.

However, a user or application must understand the contents of each local database to know what to include in a query; there is no global schema to provide advice about semantics. Also, the individual databases must maintain knowledge about the other databases with which they share information. In [Ahlsen and Johannesson 1990], this knowledge takes the form of models

of the other databases, partial global schemas, a common data model, and an explicit agreement with each of these other databases. The number of local agreements and partial global schemas may be as many as $N(N-1)$, where $N$ is the number of databases. In contrast, only $2N$ mappings are required to translate between $N$ databases and a global schema in the composite approach.

### 1.1.2 Update Operations

Updating a collection of heterogeneous databases connected over a network is a problem that has not received a lot of attention. In the composite approach, updating is related to the view update problem, for which some partial solutions have been proposed. In the autonomous approach, a user must issue updates against every affected database. In both approaches, the update problem is related to the problem of maintaining consistency among replicated or logically interrelated data items in different resources.

### 1.1.3 Design and Maintenance Operations

The above query and update operations are based on a presumption of existing heterogeneous databases. There has been little or no research on the design and incremental incorporation of additional databases.

Moreover, the world often changes faster than the formal systems that attempt to model or capture (part of) the world. In these cases, the systems become outdated. Worse, they can potentially be misused, in that users may store data with different semantics in the same category. Carnot's use of Cyc, which represents a much richer and more detailed semantics of the world, can make it obvious when such abuses occur, can enforce more elaborate semantic integrity constraints, and can thereby prevent such occurrences. It will make *ad hoc* usage much more difficult and improve accuracy, as well as consistency.

The Carnot environment will have to provide a database administrator with tools to support the design process in an integrated way. Furthermore, because of the complexity and the size of the global schema, it will be desirable to specify user views and external schemas [Sheth 1988]. A user view describes user requirements that may have nothing in common with an already existing global or external schema, as constructed in earlier schema

integration efforts. In Carnot, a user view represents additional information attached to and integrated with the global schema that can be useful for the schema integration process and for development of the transaction (query and update) processor. An external schema describes a subschema of a global schema related to a particular class of users and/or applications performing a set of activities.

## 1.2 Semantic Integrity and Integration Support

Semantic integration of autonomous information resources in Carnot is based on the composite approach for accessing a multiresource environment. This approach provides application tasks with a unified view, expressed as a global schema, of the individual resources. A user of Carnot will not be aware of differences among components of the environment and will issue queries and transactions against the unified view.

Rather than craft a new global schema each time a collection of information resources is to be integrated or each time a previously integrated resource is altered, Carnot uses the Cyc knowledge base as a preexisting global schema. The schemas of individual resources are then related to Cyc independently. This makes a global schema easier to construct and maintain than in previous attempts at implementing the composite approach. Using the excellent terminology for characterizing and taxonomizing federated database systems in [Sheth and Larson 1990], we can characterize the semantic layer of Carnot as shown in Figure 1.

It is important to note that there are at least the following two views of any information resource:

1. it can be viewed as an instance of a particular type of database management system (DBMS), such as an Oracle DBMS.

2. it can be viewed as a model (accurate or otherwise) for some portion of the real world.

The first of these views expresses the syntax that is appropriate for this resource. The second view expresses the semantics of the resource. In general, both of these views must be understood and used in order to access the resource successfully.

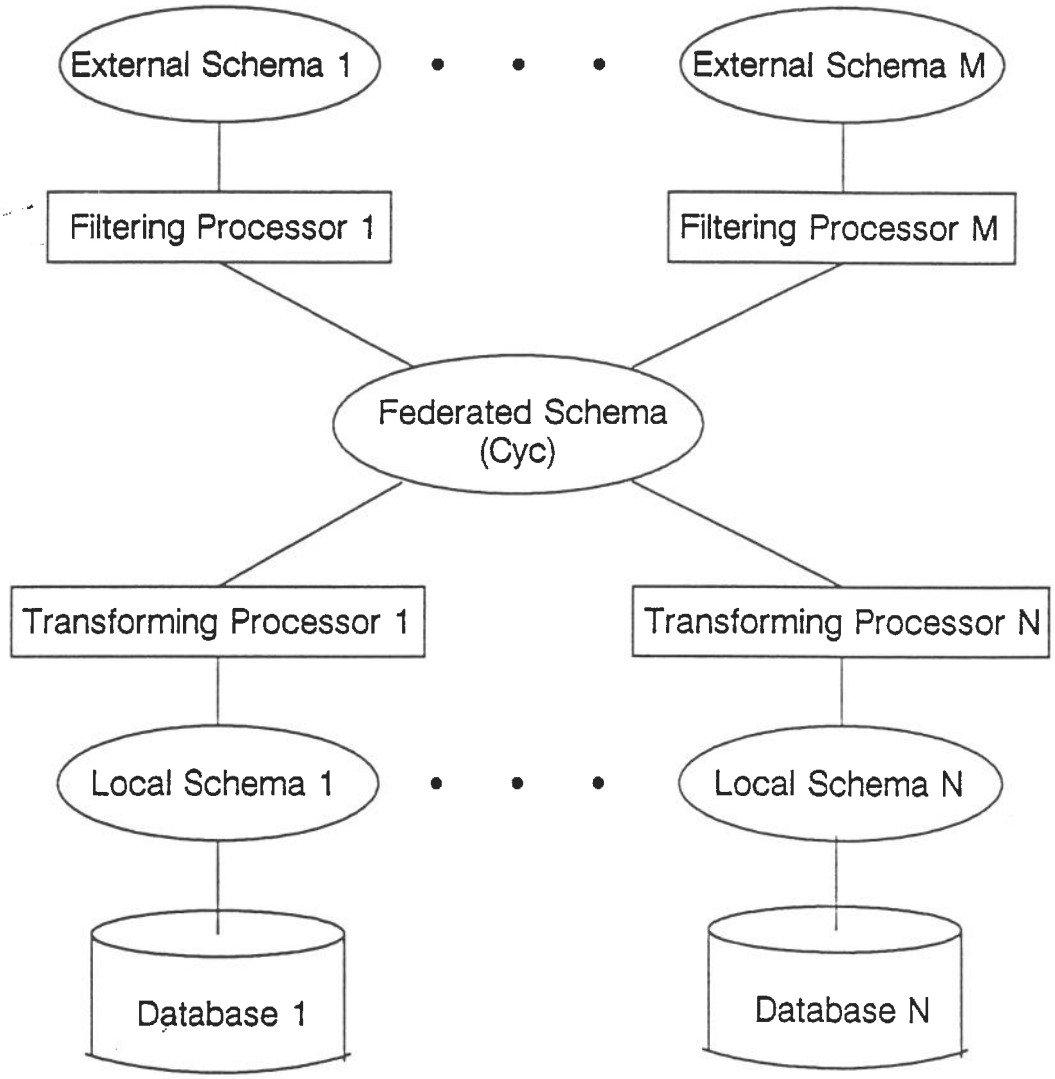



Figure 1: Architecture of the semantic layer of Carnot

We represent both of these views in Cyc for each information resource that we integrate. An advantage is that a user needs to understand only the syntax of Cyc, rather than the syntaxes of $n$ resources.

Also, most previous work on database schema integration used only a structural description of the local schemas in resolving semantic differences. We believe that successful integration requires the use of all of the information that is available about the individual information resources. This includes their data model and languages, data dictionary information, lexical definitions of the English names used for database objects, schema, the data itself, comments from the database designer or administrator, and interactive guidance from the schema integrator. We also believe that integration of resources means not only reconciling just the structural primitives of the data models supported by the resources, but also 1) reconciling integrity constraints and, more generally, rules defined on the data, 2) taking into account specifications of data usage, i.e., the programs and applications using the data, and 3) representing the semantics of the resource, i.e, the common knowledge about the resource from different points of view (e.g., the point of view of the administrator of the resource, the user of the resource, the organization to which the resource belongs, and the contribution of the resource in the heterogeneous environment—the global point of view).

The key to successful development of the composite approach is thus an ability to represent the semantics of existing information resources completely and precisely. In the next section, we digress to describe in detail the properties needed for such a representation. Section 3 then discusses the model used to represent a global schema in Cyc and provides more detail on our proposed methodology for integration. Section 4 describes our research plan.

# 2 The Requirements for Integration

We present here a preliminary analysis of the requirements for specifying the semantics of 1) individual resources and 2) collections of resources. We describe these requirements in terms of properties that qualify objects. An object can be a concept, a model, a datum, an integrity constraint, a program, etc.

## 2.1 Semantics of Individual Resources

We assume that resources may provide concepts, data, semantics, programs, languages, and end-user and administrator services. For example, a DBMS provides

- a data model describing the structural and semantic aspects of data. The semantic aspects include integrity constraints and serializability criteria.
- data, including metadata, which give a semantic interpretation of the data.
- a data definition language and a data manipulation language, allowing a set of basic query and update operations.
- tools for developing applications, loading data, controlling the physical organization of the data, controlling the use of the system, etc.
- a transaction model with its concurrent access and journal mechanisms.
- application programs acting on data through database operations.

This is a static view of a resource. We also have to consider a dynamic view, including the behavior of a resource with regard to its communication with other resources (how and when the resource responds to an external request) and the rules related to the execution of operations on shared data. In summary, a resource can be viewed as a set of objects, along with the services to manipulate the set and the rules for its use within an organization. The semantics of a resource means:

- the semantics of the objects, i.e., a conceptual representation of objects, including their definitions (types), their values, and the rules that operate on them. An object can be an application program that is executed as part of a transaction.
- the semantics of the software in terms of the services provided, e.g., a data model, one or more languages, a transaction model, etc.
- the semantics of the organization that owns the resource, i.e., the rules, defined by the organization, governing use of the resource.

Most of the work done on schema integration is based on the semantics of objects, primarily considering relationships among entities and attributes. Some of the relationships have been specified automatically by using heuristics [Souza 1986] or applying subsumption [Sheth and Gala 1989]. As in [Sheth and Gala 1989], we think that accurately comparing information (about schema, data, language, etc.) belonging to different resources, requires the real world semantics of this information, not just its represented semantics in the resource. This justifies in part the use of the Cyc knowledge base, especially its ontology and it slots hierarchy to do schema integration.

Further, we believe that attribute definition, entity definition, relation definition, record definition, etc., should be considered at the same level of importance. Every definition (a class `Person`, an attribute `age`) and every object `(JohnSmith, 35)` must be explained in terms of "real" semantic properties, i.e., the general rules it follows and the components it has. For example, each entry in a Personnel database should satisfy the definition for a real-world person (as encoded in Cyc) in that the values for its attributes should be consistent with constraints in Cyc: the `age` attribute should follow the general rule of being a piece of time, as used to represent the period of existence of a process, and satisfy the specific rules related to the definition for human ages.

Finally, little work has been done on specifying the semantics of services and organizations to facilitate query decomposition and optimization, and transaction management.

### 2.1.1 Semantics of Objects

We now present some of the properties that are useful for encoding the semantics of objects. The properties characterize objects at the description or schema level and at the value level. These properties are needed for the integration process, particularly its comparison phase (see Section 3). We use examples of objects (an individual tuple, an attribute definition, a relation definition, an integrity constraint) from the "Tour-guide" databases in Appendix A [Wang and Madnick 1989].

#### Schema-Level Properties

**name:** specifies the name of the object. A naming convention is relevant for attribute equivalence.

**domain:** specifies the type of the object; an object may have more than one type property, providing there are translation functions to convert the object from one type to another. Sometimes the domain property is related to a scale (length of a string), a range or a set of objects (domain defined by extension), or a set operation (domain defined, for example, by a cross product of domains).

**format:** specifies the number of values for that object at one time. The format is given by using one of the terms Exactly, AtLeast, Many, AtMost, or a logical conjunction of these terms.

**makes-sense-for:** specifies the permissible relationships of the object with other objects. In our example, the attribute "comments" makes sense for "FODOR-Info" and also for more general objects, such as "things" and "information."

**documentation:** defines the purpose of the object.

**synonym/homonym/antonym:** specifies the objects that are synonyms, homonyms, or antonyms of the object, respectively. There are several definitions of these. For example, synonymy could have its usual linguistic definition (such that "facility" and "amenity" from our example would be synonymous), an extended definition (such as the synonymy of two words in different languages), or a mathematical definition based on the domain or the structure of an object (e.g., "comment" and "other" are both binary relations having the same domain and range).

**structural:** specifies the composition of the object, using construct properties such as tuple, setOf, listOf, etc.

**key:** specifies that the object is all or part of a key for another object.

**integrity constraint:** specifies properties that must be held by the object to achieve resource consistency.

**side effect:** specifies rules that must be triggered when a DML operation is issued against the object.

**validation:** specifies rules that must hold/trigger when a DML operation against the object is validated/rejected.

**goal:** specifies the goal of the object. This property is used for object programs or applications, which are viewed as agents with goals. A program is defined simply as

$$< \texttt{program} - \texttt{name} >< \texttt{parameter} - \texttt{types} >< \texttt{result} - \texttt{type} >.$$

**purpose:** specifies the purpose of the object. For a program, this property gives the DDL and DML operations the program will perform. In the case of a particular program, such as a tool for interactively querying and updating a database, the property will be defined dynamically, along with the corresponding interactive program.

**participant:** specifies the list of objects in which the object participated. For example, we specify the list of programs manipulating each entity in an entity-relationship database.

Based on the above properties, we may define properties on objects belonging to different schemas. Examples of such properties are given in Section 2.2.2 and will be used to issue assertions for a global schema and translators (Section 2.3).

**Value-Level Properties** Most of the properties defined for objects at the schema level could be held by objects at the value level. The value of a program corresponds to an execution of the program. Additional properties strictly related to values are the following:

**default value:** specifies the default value for the object.

**null value:** specifies that the value of the object is a null. The value could be "does not exist," "nonapplicable," or "unknown."

**certainty:** specifies the degree of certainty of the value when it is not a null value. The certainty can be "absolute" (cannot be changed), "currently certain," or "uncertain." In a database context, the default certainty is "currently certain."

### 2.1.2 Semantics of Services

Most of the properties defined for objects can be used to qualify a set of services that a resource provides. Every service is considered to be an application, i.e., a set of programs. An application can be represented as an object with a name, a domain, a set of integrity constraints (calculated from the participants of the application), etc. For example, a relational definition language is an application whose programs are the basic definition commands. An SQL-like language is an application whose programs are the "select," "delete," "insert," and "update" programs.

### 2.1.3 Semantics of an Organization

An organization is an object with a name, a domain (an organization is always qualified by a type), a structure (an organization is composed of sub-organizations), and rules. Further, an organization can cooperate with other organizations. For each organization, the property "availability" describes when the organization responds to a request and how.

An organization is responsible for the data, programs, and applications accessible by different users. The user access rights to an application can be described using the property

**user rights:** specifies the users of the application and their rights.

Note that if we want to refine the user access right we can consider an application to be a program or a set of programs.

## 2.2 Semantics across Resources

Semantics across resources expresses how resources belonging to the Carnot environment are related. The properties listed below specify the dependencies or the links between two resources, i.e., between their objects, their services, and their organization rules provided by the resources. As we will see, some of these properties could be derived from the properties of the resources.

### 2.2.1 Relationships among Objects

#### Schema-Level Properties

**synonym:** specifies that synonymy holds between two objects from different schemas. As described above, there may be different kinds of synonym properties for an object. Let us consider first the synonym property based on the usual definition of synonymy, as follows:

$$synonym(D_1 \cup D_2) \Longleftrightarrow synonym(D_1) \cap synonym(D_2) \neq \emptyset$$

where $synonym(D_i)$ represents a set of synonymous terms from the $i$th resource. A similar definition holds for the homonym and antonym properties. Defining a synonym property is comparable to defining attribute equivalence class in [Sheth *et al.* 1988].

**common role:** specifies a relationship between concepts having something in common in terms of their role and their structural identity. This property is difficult to determine automatically. As an example, consider the attribute `category` in FODOR whose range is {inexpensive, moderate, expensive, deluxe, super deluxe} and the attribute `rating` in MASS, whose range is {$, $$, $$$, $$$$}. They are neither disjoint nor equivalent and, based on this, it is impossible to determine that `category` and `rating` may be unified. By asserting the property `common-role(category, rating)` for the global schema, the tool will suggest merging the two concepts by asking the designer to specify the necessary conversion functions.

**structural:** Most of the structural properties that can be found between two objects of schemas $S_1$ and $S_2$ arise from properties defined between the constructs of the models used by $S_1$ and $S_2$. Let us consider two representations, $R_1$ and $R_2$, of objects from the different schemas. The following properties [Batini *et al.* 1986] can be defined:

- identical: $R_1$ and $R_2$ are exactly the same. This happens when the same modeling constructs are used, same perceptions are applied, and no incoherence enters into the specification.
- equivalent: $R_1$ and $R_2$ are not exactly the same because different equivalent modeling constructs have been applied. The perceptions are still the same and coherent. There are three different types of equivalence:

1. behavioral: $R_1$ is equivalent to $R_2$, if for every instantiation of $R_1$, a corresponding instantiation of $R_2$ exists that has the same set of answers to any given query and vice versa.
2. mapping: $R_1$ and $R_2$ are equivalent if their instances can be put in one-to-one correspondence.
3. transformational: $R_1$ is equivalent to $R_2$ if $R_2$ can be obtained from $R_1$ by applying a set of atomic transformations that by definition preserve equivalence.

- compatible: $R_1$ and $R_2$ are neither identical nor equivalent. However, the modeling constructs, designer perception, and integrity constraints are not contradictory.
- incompatible: $R_1$ and $R_2$ are contradictory because of the incoherence of the specification.

The above structural properties are based on instantiations of $R_1$ and $R_2$ with their associated integrity constraints.

Comparison of schema objects is guided by their semantics and not by their syntax. To reach these properties we need to express value-level "equivalence" properties such as the equal, inclusion, overlap, and disjoint properties defined in the next section. Further, as in [Sheth *et al.* 1988], the likelihood of two high-level objects being integrated is proportional to the percentage of the "equivalence" properties defined between two low-level objects. It is clear that if the attributes of a class $C_1$ and the attributes of a class $C_2$ are all related by an "equal" property, classes $C_1$ and $C_2$ can be defined as equivalent.

**user:** Part of the semantics across resources is related to the definition of new objects and object definitions, because new concepts can arise due to the reconciliation of existing ones. For example, consider schema $S_1$ containing "Country" and schema $S_2$ containing "State." Specifying the semantics between these two schemas also means specifying a new relationship "belongs-to" between "State" and "Country."

**consistency:** Databases are said to be consistent if they satisfy all their assertions. A collection of resources with redundant and interrelated data is said to be mutually consistent if there are no conflicts among the assertions in the databases.

Starting with the assumption that databases are initially consistent, and that each transaction individually leaves the databases in a consistent state, the research literature has developed criteria to ensure that databases remain consistent in the presence of concurrently executing transactions. However, most of the work done on management of redundant and interrelated data used the conflict serializability criteria to assure full mutual consistency at all times. In the Carnot project, we want to relax this criteria along the time dimension.

To maintain mutual and global consistency of the objects in the Carnot environment, a facility provided to execute an application on a site at some time $t$ must know how to translate an operation against a datum into one or $n$ operations at a time $t$ or $t'$, $(t' > t)$, against related data stored in other resources of the distributed environment. The constraints specified among resources, along with other kinds of properties, will be used by the transaction management system of Carnot to manage consistency, i.e., to perform a transaction decomposition that determines for a given resource what other associated updates should be applied at which other resources to maintain consistency. This aspect specifies how, when, and in which order the associated updates are applied on different resources. A user update and the associated updates may be executed within a single transaction or as separate transactions. If the updates are executed as separate transactions, the associated updates may be executed as soon as possible or at a later time.

Constraints related to consistency express the way applications 1) will use the integrated view, or 2) already use objects of different resources, e.g., an update of object X of resource $R_1$ always implies an update of object Y of resource $R_2$. This kind of rule is encoded either in application programs, printed documents, or manuals.

The following properties are used to express consistency rules for updating:

- eventual consistency: this criterion states that interrelated data have to be consistent at some point in time, but may not be consistent until then. The point in time could be an interval (e.g., after one hour), an explicit time (e.g., 5 p.m. every Friday), or an

event (e.g., when a specified transaction terminates successfully).

For example, assume a service is performed for a customer. The Customer object in resource $R_1$ has been modified to take into account the realized service. This update is not posted to resource $R_2$, which is used for customer billing (an application program of the resource), until the next billing cycle begins (the event "when executing program Billing"). In the integrated schema we will have only the Customer object and objects $R_1$ and $R_2$ representing the resources.

To express eventual consistency between the two resources the following expression could be defined:

```
Consistency(R1{Customer}, R2, when_run(R2, Billing))
```

- lagging consistency: the data in one resource may be current, while related data in other resources may not be up-to-date. However, updates of data are always propagated to the related data. Some copies of data may always lag behind more current copies, but if external updates stop, the related data become consistent. Let us assume four copies of the same data, $D$, managed by four resources. The four schemas of the resources are represented in Cyc, and the parts representing the common data are mapped to the same units of the Cyc global schema. The global schema also has objects, say $R_1$, $R_2$, $R_3$, $R_4$, representing the resources. The following expression listing the resources in order of consistency could be used to indicate that the resource listed first is the most consistent for data $D$:

  ```
  laggingConsistency(R3{D}, R2{D}, R4{D}, R1{D})
  ```

  This is useful to direct queries to the most consistent data.

- periodic consistency: this criterion states that updates of interrelated data occur at specified periodic intervals.

  Both periodic and lagging consistency can be seen as special cases of eventual consistency. We may also consider mutual consistency along the space dimension and specify "how far" related data may be allowed to diverge before mutual consistency must be restored.

**Value-Level Properties** Here the term "object" means extensions of entities, relations, classes, etc., and values of single or multivalued attributes.

**group:** specifies a relationship between objects by grouping them together. For example, group({color cable TV, cable TV, color TV w/o cable, TV} 1) associates the different interpretations of TV. Therefore, any object using the elements of the group can be reconciled. The group property defines a sort of synonymy relation.

**level:** specifies a relationship among objects of the same group by differentiating some of them. This property introduces a level of granularity among the objects of the group. For example, level(TV, 1, 2) indicates that the concept (object) "TV" has the level 2 in the group 1. This property, together with the group property, allows integration of objects (definitions) using elements of a group.

**credibility:** is used to give more credibility to the value (or any of its components) of an object in a schema in comparison with the value of the "same" object in another schema. For example, the AAA relation indicates "color TV w/o cable" as a facility for the Logan Airport Hilton, but the MASS relation reports that "cable TV" is available for the Logan Airport Hilton. One way to resolve the contradiction is to incorporate the judgment that the component "facility:color TV w/o cable" of the value of the Logan Airport Hilton in the AAA relation is more credible than the component "rating:cable TV" of the value of the Logan Airport Hilton in the MASS relation.

**equal:** specifies an equality property between two objects, such that two "equal" objects are integrable into a single object.

**inclusion:** specifies an inclusion property between two objects. Object $E_1$ is included in object $E_2$ if $E_1$ is a partial replica of $E_2$. The two objects are integrable by defining an `isA` relationship between $E_1$ and $E_2$.

**overlap:** an overlap property holds between two objects $E_1$, $E_2$ if there is neither an equal nor an inclusion property between $E_1$ and $E_2$. The objects $E_1$ and $E_2$ are integrable into three objects, say $O_1$, $O_2$ and $O_3$. $O_1$ is the common part of $E_1$ and $E_2$, and is related to $O_2$, $O_3$ with

an `isA` relationship. Objects $O_2$ and $O_3$ are the complement of $O_1$ for getting $E_1$ and $E_2$, respectively.

**disjoint:** a disjoint property holds between two objects $E_1$, $E_2$ if they are disjoint as sets, and there is neither an inclusion nor an overlap property between them. Integration of the objects $E_1$ and $E_2$ is made based on anticipated utility of the new integrated object.

**calculation:** specifies the fact that an object is derived (by calculation) from other objects.

### 2.2.2 Relationships among Services

Relationships among services are specified by properties that relate models, languages, and tools. The variety of properties between two models $M_1$ and $M_2$ is strongly related to the constructs used by each of the models. To compare constructs $C_1$ of $M_1$ and $C_2$ of $M_2$ we first need a common representation of $C_1$ and $C_2$. Assume $R(C_1)$ and $R(C_2)$ are the representations of $C_1$ and $C_2$ in our model. We may find some properties that hold between them, such as identical, equivalent, subtype, supertype, etc. For every helpful pair of constructs, semantic translation will be expressed in terms of concept and rule "equivalence" properties.

### 2.2.3 Relationships among Organizations

These relationships are expressed by properties that define agreements between organizations concerning the accessibility of objects.

## 2.3 Schema Integration: Resolving Incompatibilities

Structural and semantic diversities of schemas to be merged cause conflicts and incompatibilities. In Sections 2.1 and 2.2 we list properties that apply to objects of a resource or among objects of different resources. The properties are used by a glue mechanism to yield a global schema from two representations $R_1$ and $R_2$ of two distinct schemas. By using these properties we may resolve some incompatibilities. But sometimes it is impossible to suggest to a designer exact assertions for the integration of $R_1$ and $R_2$. Therefore, merging must be explicitly specified by the designer by asserting new properties

and/or rules. We next illustrate some incompatibilities or conflicts that can arise during the integration process of two schemas [Batini *et al.* 1986].

### 2.3.1 Perspective Conflicts

Incompatibilities arise from different perspectives of the same object. Consider the relationship between employees and their departments in two schemas, $S_1$ and $S_2$. In $S_1$, `Employee` is related to `Department`, whereas in $S_2$, `Employee` is related to `Project`, which in turn is related to `Department`. The integration process may propose to collect information of $S_2$ as it contains information of $S_1$. We think that the global schema must also contain the fact that `Employee` and `Department` are coming from both schemas and that `Project` is coming from the second schema. This kind of information can be used to build and manage mapping rules, and also to introduce some "highlights" in the presentation of the global schema to the user (human interface aspects).

### 2.3.2 Structural Conflicts

Incompatibilities can arise when two schemas use different representations to describe the same object. If representations $R_1$ and $R_2$ are given for the same real-world concept, the global schema should introduce a representation that merges $R_1$ and $R_2$. For example, `Publisher` can be viewed as an attribute of entity `Book` in one schema and as a function defining the relationship between entities `Publisher` and `Book` in another schema. The global representation should be a unit `Publisher` that incorporates both attribute and relationship meanings.

The synonym, antonym, and homonym properties can be used to resolve name conflicts when the same name is used for different objects, or the same "real world" objects are described by two or more names.

Conflicts can also arise when two equivalent objects have different "definition" properties (domain property, format property, documentation property, key property, constraint property, etc.). In case of a behavioral conflict associated with two equivalent objects, the designer has to explicitly reconcile the incompatible constraint or policy properties.

### 2.3.3 Value Conflicts

Conflicts may arise from the data themselves. For example, the home address of a student may be listed as Austin in a University of Texas database and as Houston in a voting registrar's database.

# 3 Schema Integration: A Proposal

Our proposed research is being conducted in accordance with the following principles:

- Existing data should not have to be modified in order to achieve integration.
- Existing data should remain in place, and should not have to migrate.
- Existing applications should not have to be modified, unless it is desirable for them to access new or additional information resources.
- Users should not have to adopt a new language for communicating with the resultant integrated system, unless they are accessing new or additional information resources.
- Resources should be able to be integrated independently, and the mappings that result from this integration should not have to change when additional resources are integrated.

The work of most researchers that we have surveyed adheres to all but the last of these principles. We satisfy this last principle by basing our proposed composite approach on an existing global schema, provided by Cyc, rather than crafting a new global schema for each collection of information resources to be integrated. Information resources are related to this global schema—not directly to each other—with the relationships expressed in terms of mappings between each individual schema and the global schema. As a result, the relationships can be constructed independently; they do not have to be altered when other resources are related in the future. An advantage of having a global model and language is that we will need only $2N$ mapping functions for $N$ resources that we wish to integrate in the Carnot environment, instead of $N(N-1)$ functions in an approach where there is no common model and

language. Another advantage is that an update of a local schema will be propagated only to the global schema, and only one mapping function will have to be recalculated. We believe that this calculation will typically involve only a few rules of the mapping.

In accordance with the above principles and methodology, we will develop a Semantic Integrity and Integration Support (SIIS) tool. SIIS will automate and enable the implementation and subsequent use of integrated information resources. It will provide capabilities for 1) integrating local schemas, which represent the information resources, with a global schema, 2) managing such a global schema and its associated language, and 3) processing update and query transactions. It will establish a global schema/view capturing the explicit semantics represented in each of the component resources and the implicit semantics perceived by users.

We agree with [Sheth and Gala 1989] that the integration process cannot be performed automatically using current technology. However, we can provide a tool that *interactively* assists a designer who wants to integrate a resource into the Carnot environment. The SIIS tool will interact with the user to build the global schema and the requisite mapping rules. Figure 2 illustrates how this will occur.

Further, the SIIS tool will provide functionalities to correctly and efficiently execute multiresource queries and updates. Some of the requirements for this execution are the following [Landers and Rosenberg 1982]:

- transforming a query expressed in the user's global query language into a set of subqueries expressed in the different languages supported by the local DBMSs.
- formulating an efficient plan for executing a sequence of subqueries and data movement steps.
- implementing an efficient program for accessing the data at the local site.
- moving the results of subqueries among local sites.
- resolving incompatibilities between the retrieved data (differences in data types and names).

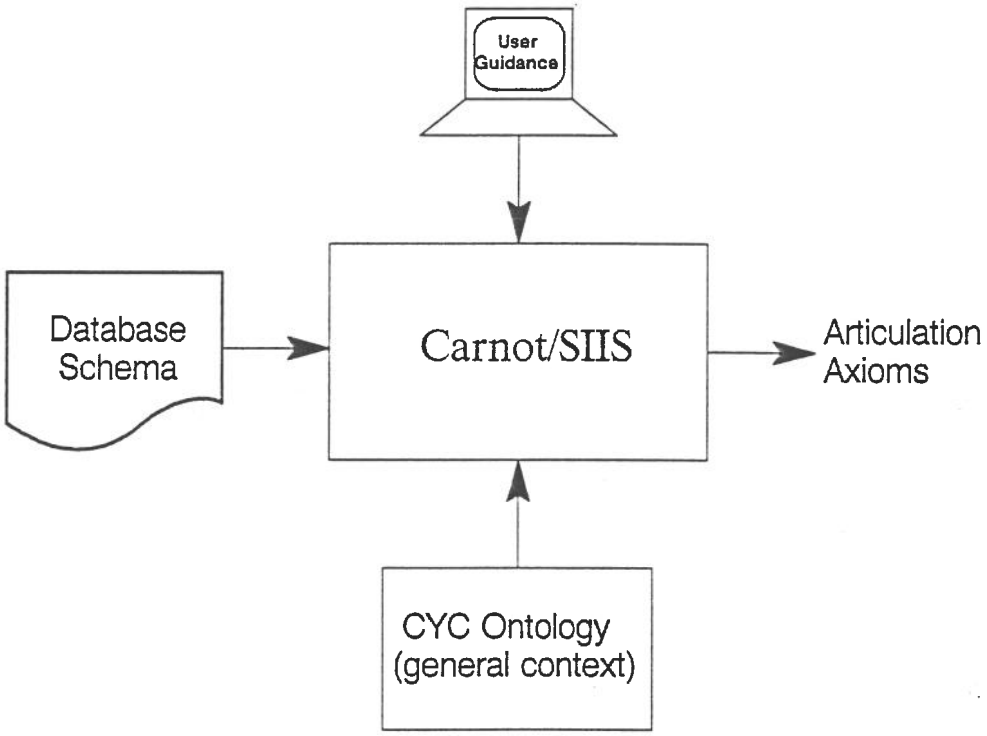



Figure 2: Interactive development of mapping rules (articulation axioms) during the schema integration phase

- resolving inconsistency in copies of the same information or in related information.
- combining the data into a single answer.

Figure 3 overviews the transaction processing required.

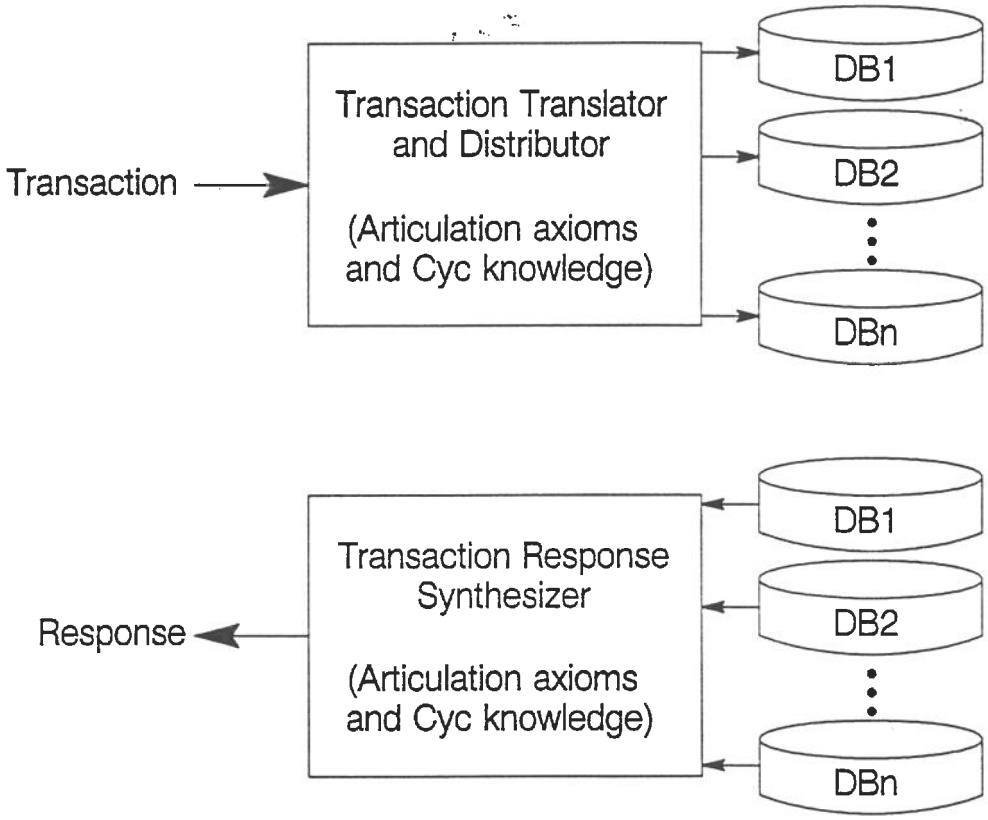



Figure 3: Transaction processing with an integrated schema

## 3.1 Methodology for Schema Integration

Any methodology for schema integration can be considered as a mixture of the following four phases [Batini *et al.* 1986]:

1. Preintegration governs the choice of schemas to be integrated, the order of integration, and a possible assignment of preferences to entire schemas or portions of schemas. This implies a set of integration policy rules, plus assertions among views.

2. Comparison of the schemas: schemas are analyzed and compared to determine the correspondences among concepts and detect possible conflicts. New properties may be discovered while comparing schemas.

3. Conforming the schemas: once conflicts are detected, an effort is made to resolve them so that the merging of various schemas is possible. Automatic conflict resolution is generally not feasible, and interaction with designers and users is typically required before compromises can be achieved. The designer must understand the semantic relationships among the concepts involved in the conflicts before choosing a solution from those proposed by an integration tool or making assertions that may resolve the conflict. The assertions must be checked for consistency so that no group of assertions leads to a contradiction.

4. Merging and restructuring: at this point the schemas are ready to be superimposed or reconciled into a global schema.

Once constructed, a global schema must satisfy the following criteria:

**completeness and correctness** —to achieve completeness, the designer has to conclude the analysis and maybe add properties or objects to the schema.

**minimality** —in most methodologies, the objective of minimality is to discover and eliminate redundancies.

**understandability** —additional schema transformations are often needed to improve understandability.

## 3.2 Carnot's Approach for Schema Integration

We will follow the above methodology in Carnot, with the following refinements: 1) we will use the Cyc knowledge base as a preexisting global schema, and 2) we will integrate each schema with Cyc independently. The knowledge

already in Cyc offers a good platform for managing the properties described in Section 2. Further, Cyc is based on a rich semantic data model, which provides a large set of mechanisms to represent a global schema. Cyc's ontology is the most stable view available of the domains of the resources to be integrated into the Carnot environment.

All four phases of the schema integration methodology are strongly influenced by the data model chosen to represent the conceptual local schemas. A simple data model has an advantage in the conforming and merging activities. On the other hand, a simpler model constitutes a weaker tool in the hands of the designer in discovering similarities, dissimilarities, or incompatibilities. A model with a rich set of type and abstraction mechanisms has the advantage of representing predefined groupings of concepts and allowing comparisons at a high level of abstraction. Further, such a model may provide concepts to specify static properties as well as dynamic properties of objects. The Cyc data model, with our representation of database theory concepts in it, provides these capabilities.

### 3.2.1 Preintegration

The preintegration phase of the integration methodology represents the schema of an information resource in the formalism of the global model, i.e., in the Cyc knowledge representation language. The representation will consist of a set of Cyc units (frames with slots). The units will reside in a separate Cyc context (microtheory) [Guha 1990] that is created for the schema. These units will be instances of more general units describing the data model used by the schema. For example, if we represent an instance of a relational schema, we will have units of types `Relation` and `DatabaseAttribute`.

The information or knowledge to be represented can be structured as in [Goldfine and Konig 1988] or [Thompson 1990]. Therefore, we will define a set of Cyc units to represent the characteristics of local resources and their related data. A first approach is to define Cyc units corresponding to the concepts of application, attribute, entity, relation, record, file, tuple, DDL and DML command, program, etc. Every local schema can be represented as instances of these units belonging to a local context. Each component resource is a subtype of a more generic component type (relational DBMS, object-oriented DBMS, hierarchical DBMS, file manager, etc.). The component types are similar to the metacontexts of [Ioannidis and Livny 1989], in that

they group together schemas that are expressed in the same formalism. New specializations of component types may be defined during preintegration, if the model and the language(s) of the resource involved in the preintegration process are not known to Cyc.

### 3.2.2 Integration

The preintegration process produces a Cyc model, contained in its own context, for an information resource. The integration process then builds a mapping between the Cyc model and the base context in Cyc. The base context contains Cyc's global ontology and constitutes the global schema. The mapping consists of articulation axioms [Guha 1990] that define correspondences between concepts in the two contexts. The axioms are based on a common representation of the semantics. Specifically, the axioms encode correspondences between the semantics of the information resource's domain and the Cyc ontology, and between its language and the Cyc language. To define these axioms we will need to use the properties of Section 2 and, based on these properties, encode rules that resolve ambiguities and incompatibilities.

Direct comparison of two information resource schemas (step 2 in Batini's methodology) will not be necessary. However, the Cyc model for a schema will be compared to units in the base context. If there are no units in the base context corresponding to ones in the model, then they will have to be created. Integration is thus an interactive process. The user may also have to assert additional properties (semantics) about the local schema and its model in Cyc. If integration of a concept yields more than one articulation axiom, the administrator of the local schema will be asked to provide additional information in order for one axiom to be chosen. Note that it may be interesting to maintain different interpretations of a concept and choose one default interpretation. If the concept is ever updated, further integration would be facilitated.

## 3.3 Using the Global Schema

It is obvious that manipulation of the global schema is tied to the use of the Cyc language. However, we do not want a data administrator of a database, a programmer, or even an end user to have to learn a new query/update language. There is a need to have a friendly interface for the representation

and the manipulation of a global schema. We will develop a graphical entity-relationship representation of the global schema and an intelligent interface for specifying queries [Weishar and Kerschberg 1989]. There is also a need for a tool for updating the global schema (adding/deleting semantics and concepts) and designing an external schema. We think that the notion of context in Cyc can be used to express the external schema.

The entity-relationship view of a global schema will be more limited, but more comprehensible, than the Cyc schema. The user will issue queries and updates against that view. Every query will then be translated into an expression in the Cyc language and processed by Cyc in order to get queries against local resources.

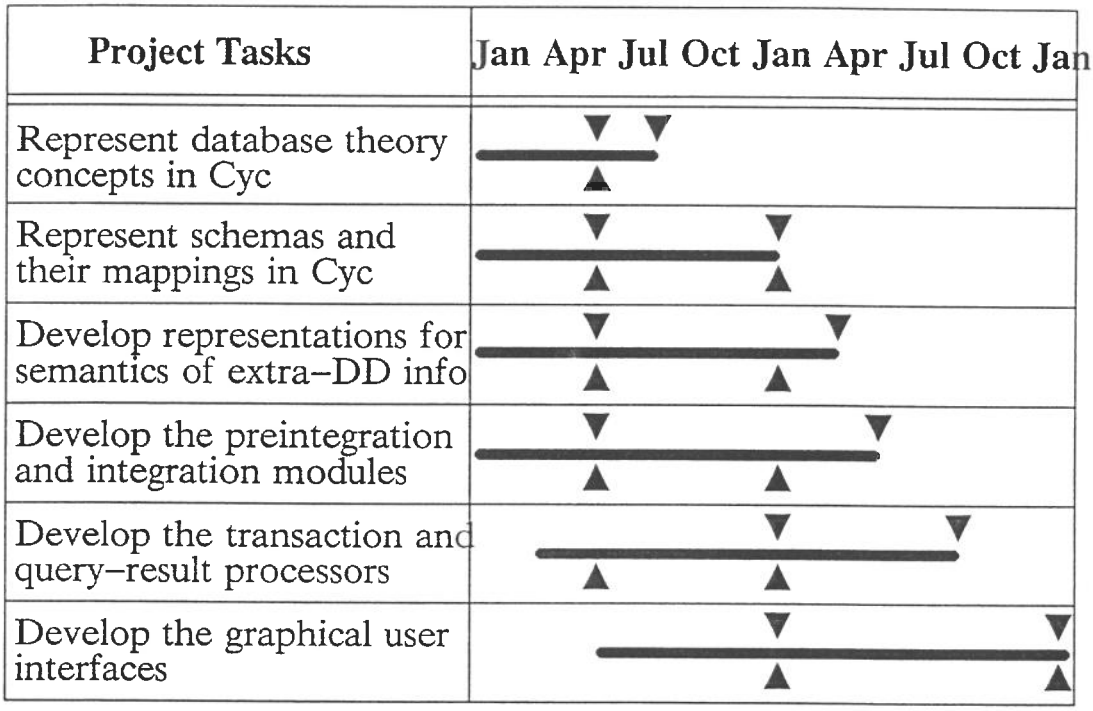
To achieve complete transparency of heterogeneity, we would prefer to provide each user with a customized perspective of the global schema, i.e., the global schema presented in the formalism of the model used by the system the user is accustomed to manipulating. Such a scenario is not easy to support. By introducing some of the properties of the autonomous approach, such as mapping or approximation rules between different resource types and the use of knowledge and metaknowledge to represent local schemas and resources (model and languages), we will make progress towards such transparency.

# 4 Research Plan

Earlier sections of this proposal have described the features and capabilities needed by a system that integrates the semantics of disparate information resources. In this section, we describe our plan for developing and implementing these features. The resultant system will support 1) the integration of schemas, 2) the management of the resulting global schema and its views, and 3) the use—at the semantic level—of the integrated information system. A summary of the schedule and milestones for the research plan is shown in Figure 4.

## 4.1 Schema Integration Support

Our plan for constructing an interactive tool that enables the integration of information resources at the semantic level has three major tasks.




Deliverables: ▼ indicates software prototype; ▲ indicates technical report or specification

Figure 4: Project schedule for January 1991 to January 1993

**Preintegration of local schemas.** We will first represent concepts of database theory in Cyc. These include types of data models, such as relational, hierarchical, and entity-relationship, and components of database systems, such as entities, objects, and relational tables. The representation of a model includes taking into account the properties of Section 2 and the IRDS standard [Goldfine and Konig 1988] in order to represent the maximum semantics about a schema and to automate, as much as possible, the definition of articulation axioms. Second, we will represent the semantics of attribute domains and integrity constraints in Cyc; these are often not specified in database schemas, although the IRDS makes a provision for them. Third, we will construct a tool that can translate a source schema expressed in some formalism of a corporate logical data model into an almost equivalent schema in the Cyc model. The corporate logical data model defines the data exchange format between Cyc and a resource.

**Integration of local schemas.** In this task, we will construct a tool that interactively builds articulation axioms for a given database using 1) the Cyc units that represent the database schema, 2) general knowledge about database schemas (encoded as Cyc units), and 3) string matches between the names of units in Cyc's general context and the names of units in the microtheory for the database schema. This task will be aided by the ability to use database semantics not present in its schema, e.g., integrity constraints, extensional information, and views of this schema in existing application programs. There will also be a means to explain decisions taken by either the tool or the administrator during the integration process.

**Interface aspects.** In this task we will develop interfaces between the integration tool and the system administrator. Specifically, we propose to

- provide a graphical entity-relationship representation of the global schema.
- provide a way to express graphical queries and updates against the global schema.
- provide a way to merge results from several databases into a coherent response, presented using the same notation as those of the global schema.

- provide a tool to help a database administrator build an interface translator between the Cyc language and the data manipulation language of the resource being managed.

## 4.2 Global Schema Management

In addition to the above tasks, we will develop the following capabilities to aid the process of updating and maintaining a global schema:

1. an ability to translate from the global schema definition to expressions of the data-definition language of a local resource (assuming the designer updating the global schema indicates which resources may be concerned with the updates).

2. an ability to integrate eventual physical design choices for the new local schema obtained after the above updating process.

3. an ability to design external schemas, i.e., subschemas of the global schema.

## 4.3 Schema Integrity Support

This task concerns the processing of transactions (query and update) created by the interfaces tools or application programs. Our plan for constructing a processor that translates a query/update against a global schema to queries/updates against local schemas has the following tasks:

1. Write a specification for and then prototype the transaction (query and update) processor that applies articulation axioms to transactions in order to distribute the transactions to appropriate databases. The transaction processor might be used off-line to generate scripts that are then executed on-line when requested. The processing of updates must consider issues of schema integrity.

2. Write a specification for the result synthesizer that uses the articulation axioms (in reverse) to merge results from several databases into a coherent response, written in the syntax of the user's query. This might also be used off-line to generate scripts for on-line execution.

3. Implement the Cyc⟷database transaction processing module.

Note that we will also define the data manipulation language used to write such transactions.

### 4.4 Evaluation

We will represent and integrate several databases with different, but overlapping, data coverage of a domain and with different data models. We will then issue query and update operations against the global schema, which will be translated and applied to these databases. If integration is successful, as we anticipate, the operations will execute correctly on each database and results will be merged coherently and returned.

## References


[Ahlsen and Johannesson 1990] Matts Ahlsen and Paul Johannesson, "Contracts in Database Federations," *International Working Conference on Cooperating Knowledge Based Systems*, University of Keele, Keele, Staffs, England, October 1990, pp. 108–111.

[Batini *et al.* 1986] C. Batini, M. Lenzerini, and S. B. Navathe, "A Comparative Analysis of Methodologies for Database Schema Integration," *ACM Computing Surveys*, Vol. 18, No. 4, December 1986, pp. 323–364.

[Beck *et al.* 1989] Howard W. Beck, Sunit K. Gala and Shamkant B. Navathe, "Classification as a Query Processing Technique in the CANDIDE Semantic Data Model," *Proceedings of Fifth International Conference on Data Engineering*, Los Angeles, CA, February 1989.

[Bell 1989] Jean L. Bell, "Research Issues for Rules Management in a Heterogeneous Database Environment," *NSF Workshop on Heterogeneous Databases*, December 1989.

[Buneman *et al.* 1990] O. P. Buneman, S. B. Davidson, and A. Watters, "Querying Independent Databases," *Information Sciences*, Vol. 52, December 1990, pp. 1–34.

[Goldfine and Konig 1988] Alan Goldfine and Patricia Konig, "The Information Resource Dictionary System," *American National Standard for Information Systems*, draft 1988.

[Guha 1990] R. V. Guha, "Micro-theories and Contexts in Cyc Part I: Basic Issues," MCC Technical Report Number ACT-CYC-129-90, Microelectronics and Computer Technology Corporation, Austin, TX, June 1990.

[Heimbigner and McLeod 1985] Dennis Heimbigner and Dennis McLeod, "A Federated Architecture for Information Management," *ACM Transactions on Office Information Systems*, Vol. 3, No. 3, pp. 253–278, July 1985.

[Ioannidis and Livny 1989] Yannis E. Ioannidis and Miron Livny, "Data model mapper generators in observation DBMSs," *NSF Workshop on Heterogeneous Databases*, December 1989.

[Kim *et al.* 1989] Won Kim *et al.*, "Features of the ORION Object-Oriented Database System," in *Object-Oriented Concepts, Applications, and Databases*, W. Kim and F. Lochovsky, eds., Addison-Wesley Publishing Co., Reading, MA, 1989.

[King and McLeod 1985] Roger King and Dennis McLeod, "A Database Design Methodology and Tool for Information Systems," *ACM Transactions on Office Information Systems*, Vol. 3, No. 1, January 1985.

[Krishnamurthy *et al.* 1988] Vishu Krishnamurthy, Stanley Y.W. Su, Herman Lam, Mary Mitchele, and Ed Barkmeyer, "IMDAS - An integrated manufacturing data administration system," *Data and Knowledge Engineering*, Vol. 3, No. 2, September 1988, pp. 109–131.

[Landers and Rosenberg 1982] T. Landers and R. L. Rosenberg, "An Overview of Multibase," *Distributed Data Bases*, H.J. Schneider (editor), North-Holland Publishing Company, 1982.

[Lenat and Guha 1989] Doug Lenat and R.V. Guha, *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*, Addison-Wesley Publishing Company, Inc., Reading, MA, 1990.

[Litwin *et al.* 1990] Witold Litwin, Leo Mark, and Nick Roussopoulos, "Interoperability of Multiple Autonomous Databases," *ACM Computing Surveys*, Vol. 22, No. 3, September 1990, pp. 267–296.

[Navathe *et al.* 1989] S.B. Navathe, S.K. Gala, S. Geum, A.K. Kamath, A. Krisknaswamy, A. Savasere, and W.K. Whang, "Federated Architecture for Heterogeneous Information Systems," *NSF Workshop on Heterogeneous Databases*, December 1989.

[Savasere *et al.* 1990] Ashok Savasere, Amit Sheth, Sunit Gala, Shamkant Navathe, and Howard Marcus, "On Applying Classification to Schema Integration," Technical Report No. TM-STS-017557, Bellcore, Piscataway, NJ, September 1990.

[Sheth 1988] Amit P. Sheth, "Building Federated Database Systems," *Distributed Computing Technical Committee Newsletter (Quarterly)*, Vol. 10, No. 2, November 1988.

[Sheth *et al.* 1988] Amit P. Sheth, J. A. Larson, A. Cornelio, and S. B. Navathe, "A Tool for Integrating Conceptual Schemas and User Views," *Proceedings of the Fourth International Conference on Data Engineering*, Los Angeles, CA, February 1988.

[Sheth and Gala 1989] Amit P. Sheth and Sunit K. Gala, "Attribute Relationships: An Impediment in Automating Schema Integration," *NSF Workshop on Heterogeneous Databases*, December 1989.

[Sheth and Larson 1990] Amit P. Sheth and James A. Larson, "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases," *ACM Computing Surveys*, Vol. 22, No. 3, September 1990, pp. 183–236.

[Sheth and Rusinkiewicz 1990] Amit Sheth and Marek Rusinkiewicz, "Management of Interdependent Data: Specifying Dependency and Consistency Requirements," Bellcore Technical Report, 1990.

[Siegel and Madnick 1989] M. Siegel and S. E. Madnick, "Schema Integration Using Metadata," *NSF Workshop on Heterogeneous Databases*, December 1989.

[Souza 1986] J. de Souza, "SIS—A Schema Integration System," *Proceedings of the BNCOD5 Conference*, 1986, pp. 167–185.

[Thompson 1990] A. K. Thompson, "The CDIF Meta Meta Model," Institute of Engineering, Ireland, May 1990.

[Wang and Madnick 1989] Richard Wang and Stuart E. Madnick, "Facilitating Connectivity in Composite Information Systems," *Data Base*, Fall 1989.

[Weishar and Kerschberg 1989] D. J. Weishar and Larry Kerschberg, "An Intelligent Interface for Query Specification to Heterogeneous Databases (extended abstract)," *NSF Workshop on Heterogeneous Databases*, December 1989.

# A The Tour-Guide Databases

We refer herein to three database schemas, from [Wang and Madnick 1989], that describe travel information for Massachusetts. These schemas represent different information and different perspectives of common information. The three database schemas are

- a relational database schema for the AAA Tour Book, with the relations:

```
AAAInfo(name*, address, rateCode, lodgingType, phone,
        other)
AAADirection(address*, direction)
AAAFacility(name*, facility*)
AAACredit(name*, creditCard*)
AAARate(name*, season*, 1PL, 1PH, 2P1BL, 2P1BH, 2P2BL,
        2P2BH, XP, fCode)
```

- an object-oriented database schema for FODOR's New England Guide, with the following classes (using the syntax of Orion [Kim *et al.* 1989], MCC's object-oriented database management system):

```
FODORInfo(name          : String,
          address       : FODORAddress,
          comment       : String,
          location      : String,
          package       : String,
          numberOfRooms : Integer,
          occupancy     : Integer,
          category      : FODORCategory,
          phones        : (set-of FODORPhone),
          facilities    : (set-of FODORFacility),
          services      : (set-of FODORService))

FODORCategory(rate      : String)

FODORAddress(street     : String,
```

```
                city        : String,
                state       : String,
                zip         : Integer,
                country     : String)

FODORPhone(phoneNum      : Integer)

FODORFacility(facilityCode : String)

FODORService(serviceCode : String)
```

- an entity-relationship[1] database schema for The Spirit of Massachusetts 1987 (abbreviated MASS), with the following entities and relationships:

  The Entity `MASSInfo` with the attributes

  ```
  name*, address, facilityType, rating, numberOfRooms,
  other, phone, cC
  ```

  The Entity `AmenityInfo` with the attribute

  ```
  amenityCode*
  ```

[1]The entity-relationship model we consider admits attributes having as values a set of atomic values. In MASS's schema, the attributes phone and cC are multivalued.