MCC Technical Report Number ACT-OODS-127-91

Resource Integration Using an Existing Large Knowledge Base

Christine Collet, Michael N. Huhns, and Wei-Min Shen

May 1991

MCC/ACT Nonconfidential

Abstract

This report describes a method for integrating separately developed information resources to enable them to be accessed and modified coherently. The method achieves integration at the semantic level by using an existing global ontology to resolve inconsistencies. We focus on the properties used to represent the semantics of a resource, which are the key to its integration. The method is incorporated in an integration tool for assisting an administrator in integrating a resource and a transaction tool for assisting users in accessing the integrated resources. The integration tool operates in two phases: 1) preintegration, in which an information resource is represented declaratively within the global ontology, and 2) integration, in which declarative mappings between the resource and the global ontology are constructed. The mappings are used by the transaction tool to translate queries and updates written against the global ontology to be distributed to this information resource when appropriate. We describe an evaluation of our method based on the integration of three databases that have different data models (entity-relationship, relational, and object-oriented) but similar semantics for their data (i.e., the databases capture information about the same domain).

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

Contents

1 Resource Integration

The goal of the research described in this paper has been to develop methods for integrating separately developed information resources to enable them to be accessed and modified coherently. The methods 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. Ultimately, the methods will provide a user with the capability to navigate through information efficiently and transparently, update the information consistently, and write applications easily for the resulting enterprise-wide 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. The Carnot system under development at MCC will provide tools that allow development of open applications that can be tightly integrated with information stored on existing, closed systems [Cannata 1991]. The semantic services of Carnot provide facilities to specify and maintain the semantics of an organization's integrated information resources. We want a common semantic framework in which we have a global or enterprise-wide view of all the resources integrated within Carnot and where we can use the same language for communicating among the resources.

We focus here on the resource integration aspect of this framework. Our approach for achieving resource integration and for resolving semantic incompatibilities among disparate information resources is to use an existing global ontology: the Cyc knowledge base [Lenat and Guha 1990]. 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.

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 against multidatabase systems [Sheth and Larson 1990]. We describe some of these methods next.

1.1.1 Query Operations

There are, in general, two approaches for 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*, is based on a monolithic global (or 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 global schema. Through this schema, 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* [Heimbigner and McLeod 1985], the *autonomous approach* [Ahlsen and Johannesson 1990], or the *multidatabase approach* [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. [Sheth and Larson 1990] and [Ahlsen and Johannesson 1990] describe five types of autonomy that can be exploited in this approach: 1) network autonomy, in which there is no central authority for communications, 2) behavioral or execution autonomy, in which there is local control of processing, 3) design autonomy, in which a local database system can independently choose the data it manages, the data representation, and the data interpretation, 4) association autonomy, in which a local database system can freely join or leave the federation, and 5) 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 only recently received much attention [Elmagarmid *et al.* 1990, Veijainen 1990]. In the composite approach, updating is related to the view up-

date problem, for which some partial solutions have been proposed. In the federated 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.

1.2 Methodology for Resource Integration

The Carnot project supports both the composite and federated approaches to semantic integration of autonomous information resources. In this report, we focus just 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 will not be aware of differences among components of the environment and will issue queries and transactions against the unified view. Using the terminology for taxonomizing federated database systems in [Sheth and Larson 1990], we can characterize the unified semantic services of our system as shown in Figure 1.

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, we use 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.

Most of these previous attempts can be characterized as some combination of the following four activities [Batini *et al.* 1986]:

1. Preintegrating—chooses the 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. Comparing—analyzes and compares schemas to determine the correspondences among concepts and detect possible conflicts.

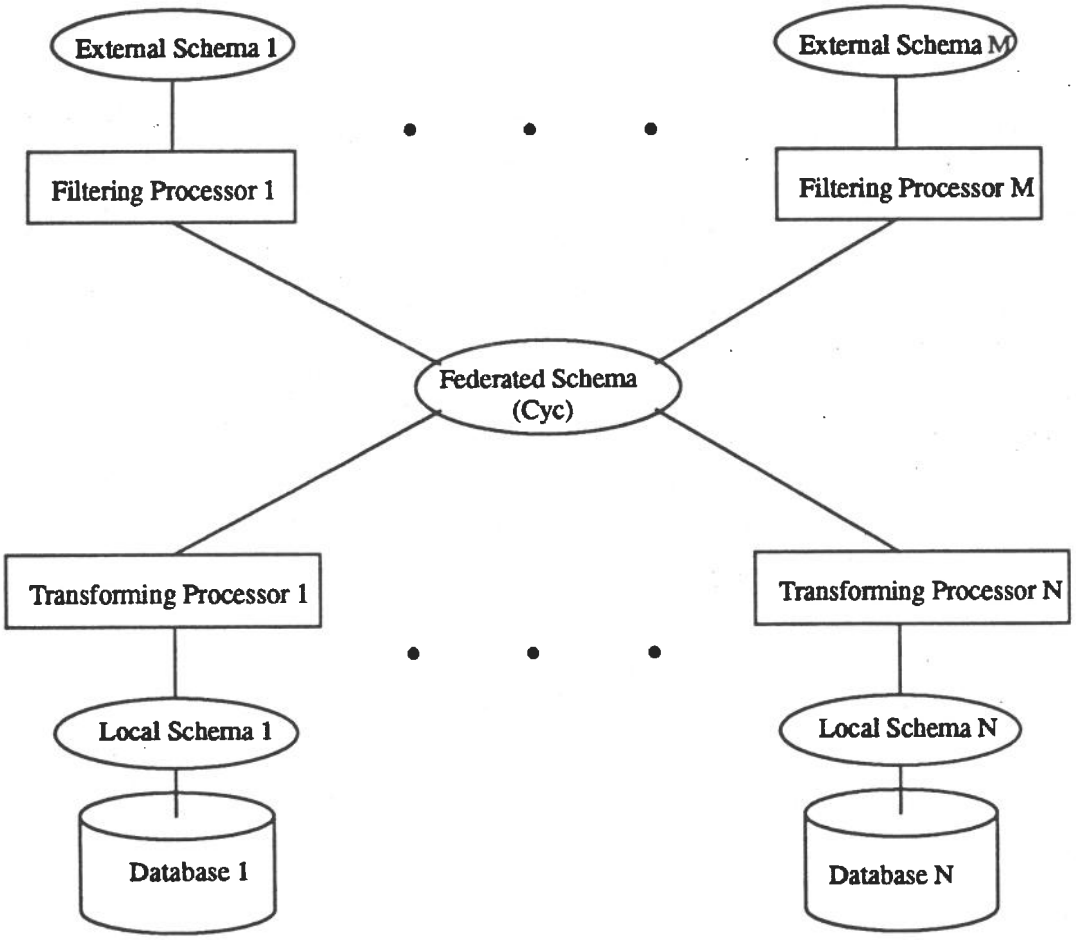


Figure 1: Architecture of the semantic services of Carnot

3. Conforming—resolves conflicts. Automatic conflict resolution is generally not feasible, and interaction with designers and users is typically required before compromises can be achieved.

4. Merging and restructuring—superimposes and reconciles schemas into a global schema.

Once constructed, the global schema should be complete, correct, minimal, and understandable.

Our methodology differs in many aspects from that described above. First, resource integration means more than just schema integration. Also, most previous work on database schema integration used only a structural description of the local schemas in resolving semantic differences. We believe that integration of resources means not only reconciling 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, 3) representing information about the resources themselves, and 4) including organizational and management information.

Therefore, a successful integration requires the use of all of the information that is available about the individual information resources. This includes:

- the schema information, i.e., the structure of the data, integrity constraints, and allowable operations;

- the resource information, i.e., a description of the services supported by the resource, such as the data model and languages, data dictionary information, lexical definitions of the names used for database objects, data itself, comments from the database designer or administrator, and interactive guidance from the schema integrator.

- the organization information, i.e., the rules defined by the organization to which the resource belongs. The rules specify the contribution of the resource in the heterogeneous environment and the characteristics of the organization, e.g., the natural language used to communicate, the management rules and the design methodology implemented.

We reiterate that schema integration is only a part of the resource integration problem.

Second, our process of integrating resources is different in that we 1) use the Cyc knowledge base as a preexisting global schema, and 2) integrate each schema with Cyc independently. The schemas are compared, conformed, and merged with the Cyc ontology, but not with each other. During semantic transaction processing, however,

Cyc will compare them to determine the most appropriate local resources to process queries and updates.

The knowledge already in Cyc offers a good platform for managing the properties described in Section 3. This knowledge base currently contains the equivalent of 50,000 entities and relationships expressed as frames and slots. Cyc not only encodes the semantics for a significant portion of human consensus reality to which a particular information resource can be related, but also provides the knowledge representation and inference mechanisms needed for expressing the relationships among information resources. It can represent dynamic properties of data via predicates attached to the slots of the frames. Further, because it is based on a rich semantic data model, it provides the large set of mechanisms necessary to construct, represent, and maintain a global schema. For maintenance, Cyc can enforce more elaborate integrity constraints and can thereby prevent violations of semantics. Cyc's ontology is the most stable view available of the domains of the resources to be integrated into an enterprise-wide information 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.

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 resource, such as a relational database management system (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. 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 and semantics of n resources.

The key aspect of our approach is thus an ability to represent the semantics of existing information resources completely and precisely. The semantics can be derived from on-line information (a data dictionary, stored data and definitions, etc), calculated, or specified explicitly by the administrator of the resource. In the next section, we analyze three example databases and identify problems that arise in representing their semantics and integrating them. We then describe in detail the properties needed for such a representation in Section 3. Section 4 presents the Cyc concepts used during a *preintegration phase* to represent aspects of an information resource: the objects and applications involved, the services provided (the data model, the transaction model, the languages, the tools, and the utilities), and its place in an organization (the usage rules for the resource) [Collet and Huhns 1991]. Section 5 describes the *integration phase*, during which the representation of the local schema in Cyc is related to the global schema (Cyc ontology). Section 6 introduces transaction processing with a resultant integrated schema. Section 7 concludes the paper and presents ideas for future research and experimentation.

2 Example Schema Integration Analysis

We use three database schemas, adapted from [Wang and Madnick 1989] and listed in Appendix A, to introduce problems arising in schema integration. The schemas are structurally and semantically different. Structurally, these schemas are from three different data models: entity-relationship, relational, and object-oriented. Semantically, these schemas contain different information and different perspectives about items common to the three databases, such as accommodations in Boston.

The three databases, each providing information about lodging, belong to different organizations. Let us assume that these organizations allow a travel service to access their databases, and that the travel service wishes to provide its customers with the maximum amount of information about lodging when planning a trip. To realize this objective, the travel service will have to integrate the local schemas of the databases in order to get a global schema through which queries can be issued by users. However, structural and semantic diversities of the schemas cause conflicts and incompatibilities. We discuss next a few of the problems that arise in integrating the example databases.

Problem 1: the schemas use different names for the same attributes. The relationship `MASSAmenity` in the MASS schema is similar to the `facility` attribute of the relation `AAAFacility` and the `facilityCode` attribute of the class `FODORFacility`. A *synonym property* could be attached to these terms specifying their equivalence, if we were trying to merge these schemas. In our method, we map each of these

independently into the preexisting Cyc slot `hasAmenities`.

Problem 2: the schemas use different names for the same attribute values. In this case, a synonym property would not resolve incompatibilities among the values of the attributes: in MASS, an `amenityCode` can have the value "6," which corresponds to the value "pool" for AAA's `facility` and the value "outdoor pool" for FODOR's `facilityCode`. [Wang and Madnick 1989] resolves this by defining group and level properties. A *group property* states that the members of the group belong to the same semantic category. For example, the values "pool," "outdoor pool," and "6" are in the same group. A *level property* encodes a notion of specificity among the objects of the group, enabling the values to be compared and ranked. A higher value indicates a more generic concept. For example, the concepts "pool" and "6" are at the same level in the same group, and thus have the same semantics. Similarly, the values {color cable TV, cable TV, color TV w/o cable, TV} can be grouped and then assigned a level property to indicate that "TV" is the most generic.

Problem 3: a query may return contradictory information from different databases. In this case, a *credibility property*, attached to the synonymous attributes, can be used to state that one information resource is to be preferred over another when they produce conflicting values for that attribute. For example, an answer to a query may return "cable TV" and "color TV without cable" for the same hotel. The contradiction is resolved by encoding the judgment that the attribute `facility` of the AAA database schema is more credible than the attribute `facilityCode` of the MASS database schema.

A practical difficulty with the use of group, level, and credibility properties is that they cannot be derived from a schema definition, and must be given by the administrator responsible for the integration of the schemas. A conceptual problem with the level property is that it provides only a linear ordering of specificity, and a partial ordering is often more appropriate. For example, the values "cable TV" and "color TV without cable" are incommensurate with respect to specificity.

Problem 4: when attributes have enumerated data types, the values may be incommensurate. For example, the four possible values for the attribute `rating` in MASS, {$, $$, $$$, or $$$$}, are difficult to relate to the five possible values for the attribute `category` in FODOR, {inexpensive, moderate, expensive, deluxe, or super deluxe}. One possible solution is to attach a *common role property* to the attributes `rating` and `category`, specifying that they have something in common, and then include a conversion function to translate the values. A better solution is to map each of these into a more specific representation. Cyc uses a numerical representation for

money, allowing maximum, minimum, and typical values, that subsumes the other two representations.

3 Properties for Semantic Specification

We present here an analysis of the requirements for specifying the semantics of both individual and collective resources. 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. Objects can be concepts, models, data, integrity constraints, application programs, etc. 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.
- 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 our use of the Cyc knowledge base, especially its ontology and it slots hierarchy.

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.

3.1 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.

3.1.1 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 "FODORInfo" and also for more general objects, such as "things" and "information."

documentation: defines the purpose of the object.

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 query or update operation is issued against the object.

validation: specifies rules that must hold (trigger) when a query (update) 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. The external behavior of a program is declaratively represented as

`<program-name> <parameter-types> <result-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.

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). Defining a synonym property is comparable to defining an attribute equivalence class, as in [Sheth *et al.* 1988].

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.

credibility: is used to state preferences about the value (or any of its components) of an object in one schema compared to the value of the "same" object in another schema.

consistency: 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. The constraints specified among resources, along with other kinds of properties, will be used by a transaction management system to manage consistency, i.e., to perform a transaction decomposition that determines for a given resource what other associated updates should be applied how, when, and at which other resources to maintain consistency. The following properties express consistency rules for updating:

- `eventualConsistency`: states that interrelated data have to be consistent at some point in time, but may not be consistent until then. For example, assume a service is performed for a customer. The Customer object in

resource R_1 is modified to take into account the realized service, but this update is not posted to resource R_2 until the next billing cycle begins (the event "when executing program Billing"), expressed as

```
eventualConsistency(R1{Customer}, R2,
                    whenRun(R2, Billing))
```

- `laggingConsistency`: the data in one resource may lag behind current data in another resource, 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 ordered list of resources indicates that the one 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.

- `periodicConsistency`: states that updates of interrelated data occur at specified periodic intervals.

3.1.2 Value-Level Properties

Most of the properties defined for objects at the schema level could be held by objects at the value level. Value-level objects are extensions of entities, relations, classes, etc., and values of single or multivalued attributes. 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," "currently certain," or "uncertain." In a database context, the default certainty is "currently certain."

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

4.1.3 DatabaseLink

The unit `DatabaseLink` is used to relate high level concepts to each other, or to relate a high level concept with an attribute. This unit summarizes the common features of a set of link definitions, i.e., definitions of relational attributes, entity-relationship attributes, class attributes (that can also be viewed as public methods of a class interface to access the state of an object of that class), or links involved in a relationship.

The `DatabaseLink` unit is a specialization of `Slot` and, therefore, any instance of `DatabaseLink` can have bookkeeping slots, the `makesSenseFor` slot, the `entryIsA` slot, the `entryFormat` slot, etc. All these slots are defined in the units generalizing `DatabaseLink` and will be used to express some of the properties of objects at the schema level from Section 3. A `Link` definition is a *slot definition* whose slot `makesSenseFor` has as value an instance of a `DatabaseObjectDefinitionType` or any of its specializations. For other properties attached to a link, such as primary key, synonym, antonym, null values, and default value, we have to define new slots for the `DatabaseLink` unit.

The `DatabaseLink` unit has the following two specializations (shown in Figure 4):

- `DatabaseAttribute`, which summarizes the common features of a set of attribute definitions and has three specializations:
  1. `RelationalAttribute` summarizes the common features of a set of definitions of relational attributes.
  2. `ERAttribute` summarizes the common features of a set of definitions of entity-relationship attributes.
  3. `OOClassField` summarizes the common features of a set of definitions of class attributes.
- `ERLink`, which summarizes the common features of a set of relationship link definitions.

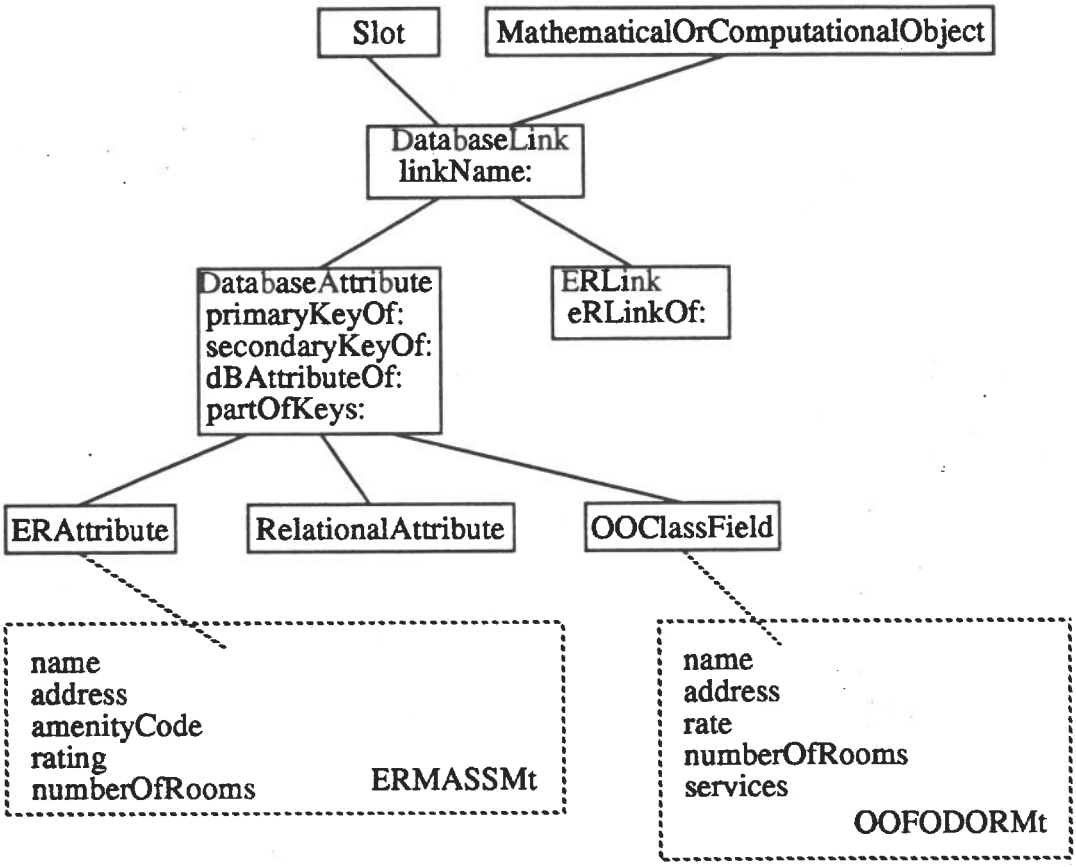
4.2 Preintegration Tool

4.2.1 Preintegration of a Data Model

This process is used when a new resource is integrated into the environment.

Input: type of the resource.

Interactive process: representation of concepts



Note: solid lines denote subset relationships, dashed lines denote element relationships, and dashed boxes denote microtheories.

Figure 4: Cyc units for representing attributes of database components

1. The type of the resource to be integrated is already represented in Cyc: the tool will propose a representation that the administrator of the resource may specialize by deleting slots, adding slots, or modifying slot definitions.
2. The type of the resource to be integrated is unknown to Cyc: the administrator must represent the data model for this resource in the Cyc knowledge base.

Output: the units for the representation of a data schema of the resource.

4.2.2 Preintegration of a Schema

Input: the schema S for a resource, expressed in the format for data exchange between Cyc and the resource.

Interactive process: Cyc representation of S

1. Define a context (microtheory) C that is an instance of `DBContext`; in C, define a unit for S that is an instance of `DatabaseSchema`.
2. For every object or concept O in S, create a corresponding unit in the C context of Cyc, according to Table 1, and link it to the unit defined for S. Parts of this operation are interactive, e.g., in representing an object method as a Cyc predicate, the system can only *suggest* the argument types for the predicate and the user must make the final choice. Notice in Table 1 that an attribute is represented by a slot in Cyc. The value of `makesSenseFor` for this slot is the object that the attribute describes. The value of `entryIsA` for this slot is assigned according to Table 2. The value of `entryFormat` is `SetTheFormat` if the attribute value is a `set-of` construct, and is `SingleEntry` otherwise.

Output: the Cyc units representing the schema.

5 Integration

The preintegration process produces a Cyc context containing a model 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 encode correspondences between the semantics of the information resource's domain and the Cyc ontology, and between its language and the Cyc language.

Table 1: Cyc Structures for Representing Database Components

Data Model Concept	Cyc Structure
Object Class	Collection
Object Instance	Individual
Object Attribute	Slot
Object Method	Predicate
ER Entity	Collection
ER Relationship	Collection
ER Attribute	Slot
Relational Table	Collection
Relational Attribute	Slot
Relational Tuple	Individual
Hierarchical Segment	Collection
Hierarchical Record	Individual
Hierarchical Field	Slot
Codasyl Record Type	Collection
Codasyl Record	Individual
Codasyl Data Item (Field)	Slot
Codasyl Set (Link)	Slot

Table 2: Cyc Structures for Representing Database Attribute Values

Attribute Value	Cyc Structure
Char(n), Char$n$, String	`#%LispString`
Num(n), Int$n$, Integer	`#%Integer`
Enumerated Type	`#%Collection` with values as instances

The integration process proceeds through two sequential phases. In the first phase, concepts from the Cyc model of the local schema being integrated are matched with appropriate concepts in the global schema. If there are no units in the global schema corresponding to ones in the local schema, then they are created. Matching is thus an interactive process. The user may also have to assert additional properties (semantics) of the local schema and its model in Cyc, utilizing the properties defined in Section 3. In the second phase, the matches are converted automatically into articulation axioms by instantiating templates for these axioms with terms from the matches.

5.1 Matching

The matching phase of integration can be considered as the dual problem to conceptual modeling for resource design (or to knowledge representation for expert system design). In conceptual modeling, the problem is: given a concept from the real world, determine how to represent and model it. In resource integration, the problem is: given a representation for a concept, determine the concept. There are several factors that affect this phase: there may be a mismatch between the local and global schemas in the depth of knowledge representing a concept, and there may be mismatches between the structures used to encode the knowledge. Specifically,

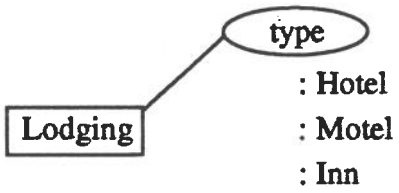
Concept Representation in the Local Schema: the local schema may have used one of three different structures to represent a concept, roughly corresponding to the primary structures of the relational, object-oriented, and entity-relationship data models. These three structures are shown by example in Figure 5.

Concept Representation in the Global Schema: the global schema may have used one of two different structures to represent a concept. In Cyc, a concept can be represented as either a category or an attribute [Lenat and Guha 1990, pp. 339ff]. These structure types are shown by example in Figure 6.

Relative Knowledge: the global schema may have more, less, or equivalent knowledge compared to a local schema. This factor applies to each concept in the local schema, rather than to the local schema as a whole. (It is interesting to compare Cyc's knowledge about swimming pools, shown in Figure 7, with that of the travel databases, described in Appendix A.)

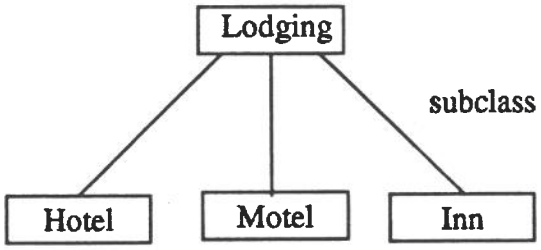
If the global schema's knowledge is more than or equivalent to that of the local schema's for some concept, then the interactive matching process described in this section will find the relevant portion of the global schema's knowledge. This knowledge will be in one of Cyc's two forms for concept representation. If the global schema has less knowledge than the local schema, then knowledge will be added to the global schema until its knowledge equals or exceeds that in the local schema. Otherwise, the global schema would be unable to model the semantics of the resource. The added knowledge, represented in terms of Cyc concepts, refines existing knowledge in Cyc. For example, if an entity in a local schema has an additional attribute compared to the corresponding entity in Cyc, then a slot is created in Cyc to represent the semantics of the attribute.

Finding correspondences between concepts in the local and global schemas is a subgraph-matching problem. Subgraph matching is based on a simple string matching between the names of units representing the database schema and the names of Cyc units. It also uses what we have called the synonymy property in Section 3. Matching



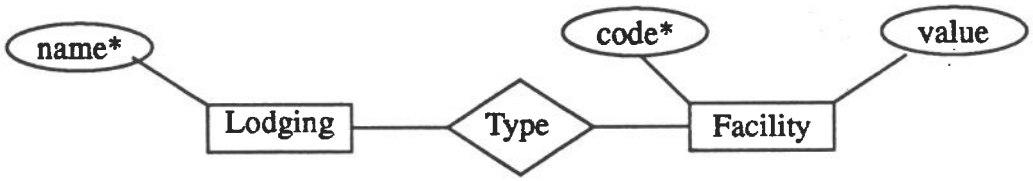
Schema concept represented as attribute

SELECT * FROM Lodging WHERE type = "Hotel"



Schema concept represented as class

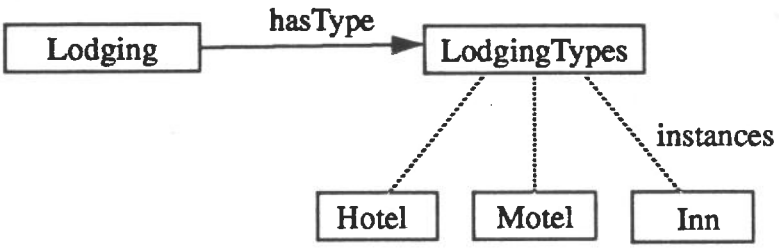
SELECT * FROM Hotel



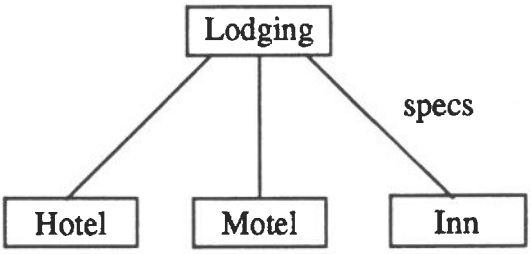
Schema concept represented as relationship

SELECT * FROM Lodging, Type, Facility WHERE
Lodging.name = Type.name AND
Type.code = Facility.code AND
Facility.value = "Hotel"

Figure 5: Three possible representations for the concept `Hotel`, each allowing a different aspect of it to be emphasized. Also shown are example SQL query forms



Cyc concept represented as a slot



Cyc concept represented as a category

Figure 6: Two Cyc representations for the same concept, each allowing different aspects of it to be emphasized

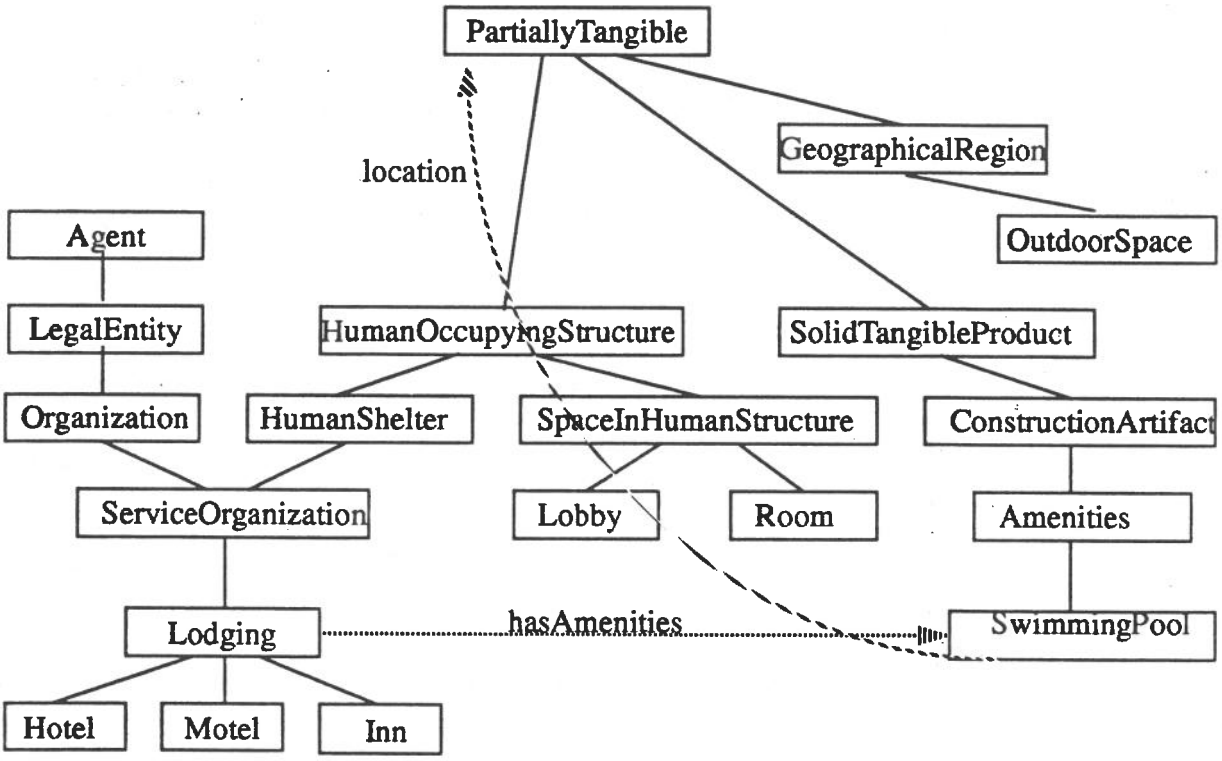


Figure 7: A small portion of Cyc's ontology concerning swimming pools. Note that this is a more detailed representation than that in the database schemas in Appendix A

begins by finding associations between attribute/link definitions and existing slots in Cyc. For example, for the MASS database, matching of the attribute definition `numberOfRooms` in the ERMASSMt context results in an association with the existing slot `numberOfRooms` in the Cyc global context.

After a few matches have been identified, either by exact string matches or by a user indicating the correct match out of a set of candidate matches, possible matches for the remaining schema concepts are greatly constrained. (Conversely, after integrating an entity or object E_i, possible matches for its attributes will be greatly constrained.) Specifically, let a_{ij}, $j = 1, 2, \ldots, n$ denote the attributes of concept E_i in a local schema. E_i is the domain of the attributes, i.e., the entity, relationship, relation, class, or object for which the a_{ij} are defined. Let s_j be the slot in Cyc that corresponds to, or matches, a_{ij}.

Observation 1 *The domain C_j of slot s_j (i.e., the value of the* `makesSenseFor` *slot of s_j) is a generalization of the concept in the global schema that matches E_i.*

For example, the domain of the attributes `numberOfRooms` and `phone` is the entity `MASSInfo`, whereas the domains of the corresponding Cyc slots `numberOfRooms` and `phoneNumber` are the units `HumanOccupyingStructure` and `Agent`, respectively. These are generalizations of the unit `Lodging` in Cyc, which is the unit whose semantics most closely corresponds to `MASSInfo`.

As we match each of the attributes of E_i, we compute the common subdomain of the domains of their corresponding slots, i.e., the intersection of the values of the `makesSenseFor` slots of these slots. The resulting common subdomains, although still generalizations of E_i, approximate it more and more closely.

Observation 2 *The "best" match for E_i is $\bigcap_{j=1}^{n} C_j$, the most general common subdomain (greatest lower bound in the generalization hierarchy) of the slot domains.*

In the above example, the most general common subdomain of `Agent` and `HumanOccupyingStructure` is `ServiceOrganization`, a generalization of `Lodging`. This would be suggested as the approximate match for `MASSInfo`. If no other attributes are matched, this would also be the *best* match that could be determined automatically for `MASSInfo`.

The greatest lower bound might not exist as a single unit in the global schema, however; it might be a set of units. In this case, a unit would be created in the base context of Cyc with the units in the set listed as its generalizations.

Unfortunately, string matching on names is too weak of a method for suggesting candidate matches. We need a mechanism, based on properties of the concept, that enables the matching process to identify one or more units close enough to the concept that articulation axioms can be written. For example, consider the integration of the

attribute **other**. The value of this attribute defines a description or a comment for an entity **MASSInfo**, so its semantics are similar to the semantics of the Cyc slot **english**. The only means we see for finding such a similar slot is by 1) accessing the value of the slot **entryIsA** of **other** to find its domain C, 2) finding and listing all of the Cyc units that have C as the value of their slot **entryIsA**, and 3) asking the administrator to choose one from the list.

5.2 Constructing Articulation Axioms

An articulation axiom is constructed for each match found. For example, the match that is found between the attribute **numberOfRooms** and the Cyc slot **numberOfRooms** results in the following axiom:

$$\texttt{isTrue(GC allInstanceOf(LODGING Lodging)} \wedge$$

$$\texttt{numberOfRooms(LODGING NUMBER))}$$

$$\Longleftrightarrow$$

$$\texttt{isTrue(ERMASSMt numberOfRooms(LODGING NUMBER))}$$

which means that the **numberOfRooms** attribute definition determines the **numberOfRooms** slot in the global schema, and vice versa. The notation **isTrue(C P)** states that **P** is true in context **C**. The context **GC** used in this latter axiom is a set of Cyc units that covers the semantic domain of the MASS database. **GC** could be the Cyc base context, or any other existing Cyc context. Similarly, the match between the Cyc collection **Lodging** and the entity **MASSInfo** yields the axiom

$$\texttt{isTrue(GC allInstanceOf(LODGING Lodging))}$$

$$\Longleftrightarrow$$

$$\texttt{isTrue(ERMASSMt allInstanceOf(LODGING MASSInfo))}$$

Articulation axiom construction is accomplished automatically by using the matches to instantiate templates for the axioms, such as the templates shown in Table 3.

A difficult example is the attribute **rating** in MASS whose value (**$**, **$$**, **$$$**, or **$$$$**) indicates the cost of the lodging described by a **MASSInfo** entity. Assume we have already integrated the FODOR schema and have a slot **category** in the global schema with the domain {inexpensive, moderate, expensive, deluxe, super deluxe}. To define an articulation axiom that reconciles **category** and the **rating** attribute, the database administrator will have to associate a common role property (a new slot) to **rating** (or its domain, i.e., the value of its **entryIsA** slot) whose value is a conversion function.

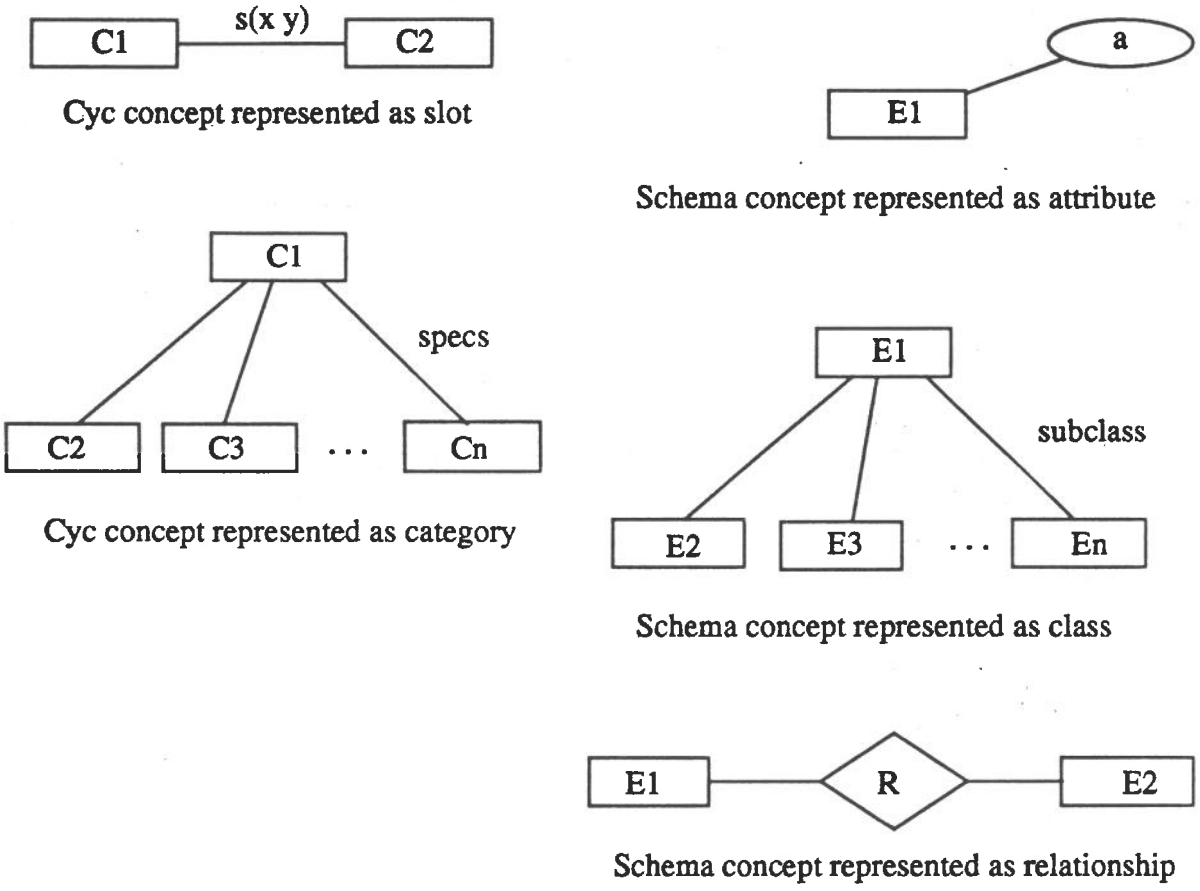


Figure 8: Possible representations in Cyc and resource schemas for the same concept, each allowing a different aspect of it to be emphasized

Table 3: Templates for Building Articulation Axioms

Cyc Concept Represented:	Schema Concept Represented:	Articulation Axiom Template
as slot	as attribute	$isTrue(GC\ aIO(x\ C_1) \iff isTrue(LC\ aIO(x\ E_1) \wedge s(x\ y)) \wedge a(x\ y))$
as slot	as class	$isTrue(GC\ aIO(x\ C_1) \iff isTrue(LC\ aIO(x\ y)) \wedge s(x\ y))$
as slot	as relationship	$isTrue(GC\ aIO(x\ C_1) \iff isTrue(LC\ aIO(x\ E_1) \wedge aIO(y\ C_2) \wedge aIO(y\ E_2) \wedge s(x\ y)) \wedge aIO(z\ R) \wedge r_1(x\ z) \wedge r_2(z\ y))$
as category	as attribute	$isTrue(GC\ aIO(x\ C_2) \iff isTrue(LC\ aIO(x\ E_1) \wedge specs(C_1\ C_2)) \wedge a(x\ E_2))$
as category	as class	$isTrue(GC\ aIO(x\ C_2) \iff isTrue(LC\ aIO(x\ E_2) \wedge specs(C_1\ C_2)) \wedge subclasses(E_1\ E_2))$
as category	as relationship	$isTrue(GC\ aIO(x\ C_2) \iff isTrue(LC\ aIO(x\ E_2) \wedge specs(C_1\ C_2)) \wedge aIO(y\ E_1) \wedge aIO(z\ R) \wedge r_1(y\ z) \wedge r_2(z\ x))$

Notes: these axioms assume that global entity C_i matches local entity E_i. *aIO* denotes `allInstanceOf`, which is the transitive closure of the `instanceOf` slot. *GC* denotes the global schema context, *LC* denotes the local schema context, and *specs* denotes Cyc's `subclass` relation. The other symbols are explained in Figure 8.

Our objective is to automate to a maximum extent the integration process and, therefore, the integration phase. However, as shown above, there is a need for information that is not contained in the metadata or data stored in the resource. Therefore, the properties specifying such information and introduced in Section 3 cannot be defined in an automated way. These properties must be added by the resource administrator during the integration process, so there is a need for an interface that allows collaborative work between the integration tool and the database administrator. Figure 9 summarizes the preintegration and integration processes.

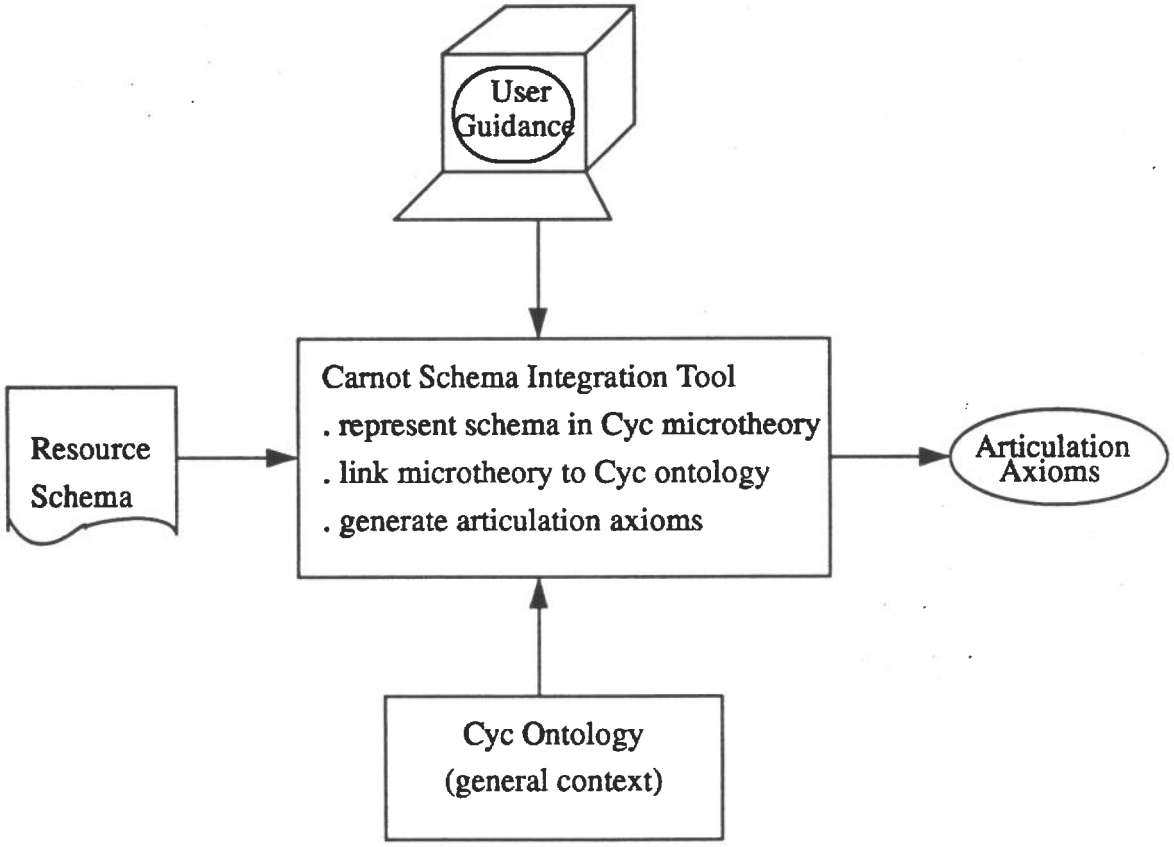


Figure 9: The preintegration and integration phases of the interactive development of semantic mapping rules for multiresource transaction processing

6 Semantic Transaction Processing

Semantic transaction processing concerns the processing of multiresource queries and updates created by interface tools or application programs. Some of the requirements for this processing are [Landers and Rosenberg 1982]:

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

In Carnot, the semantic transaction processor provides information to a distributed transaction generator in order to describe in a distributed operation language what steps are needed to query and update the data. The semantic transaction processor translates a query or update against the global schema to a set of queries or updates against local schemas by applying articulation axioms. The set of queries and updates are then sent to a distributed transaction generator, along with dependency properties among updated data, consistency requirements for these data (represented by `eventualConsistency`, `periodicConsistency`, or `laggingConsistency` properties), and information for building query results. The transaction generator is responsible for distributing the transactions to appropriate databases and managing their execution. Note that the execution of updates must consider issues of schema integrity.

After the transactions are executed, the semantic transaction processor 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. The transaction processor can also be used off-line to generate scripts that are then executed on-line when requested. Figure 10 overviews the transaction processing required, while Figures 11 and 12 show examples of using the articulation axioms in Figures 13 and 14 in the processing of queries.

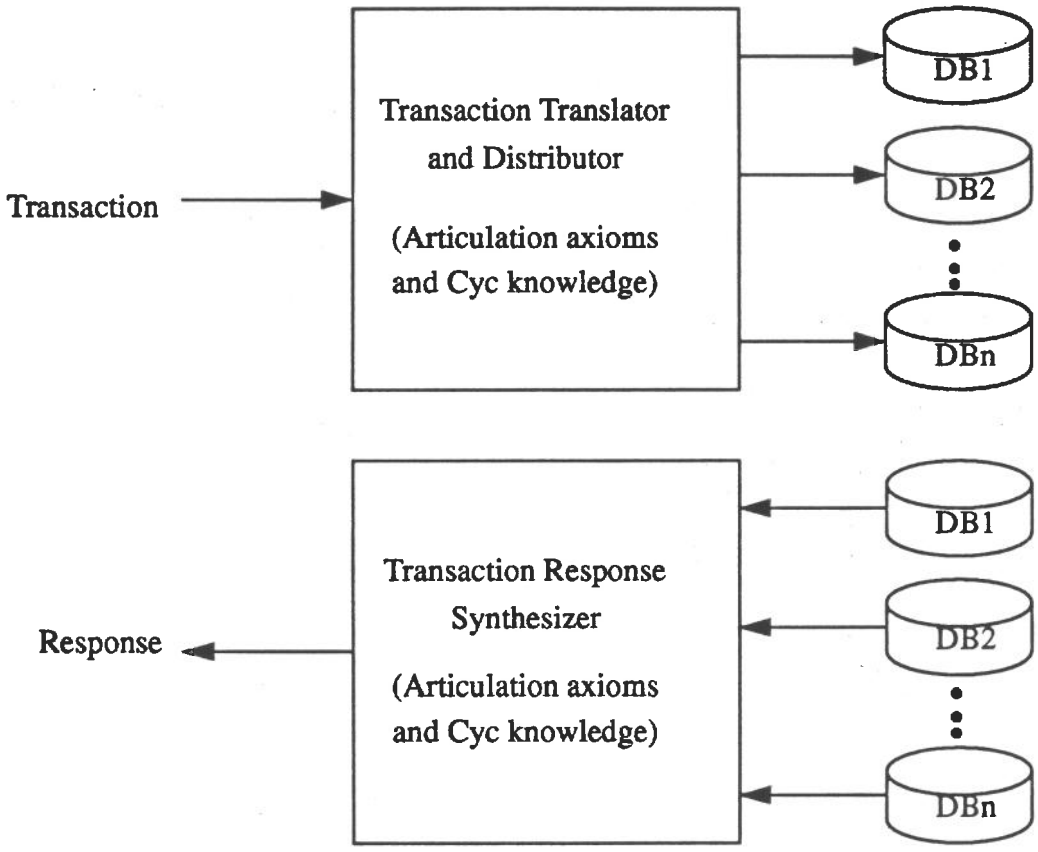


Figure 10: Semantic transaction processing with an integrated schema

SQL query requesting information about hotels:

```
SELECT * FROM Lodging
```

Query after translation to CycL:

```
(allInstanceOf LODGING Lodging)
```

Queries (in CycL) for MASS and FODOR databases after translation using articulation axioms:

```
(allInstanceOf LODGING MASSInfo)

(allInstanceOf LODGING FODORInfo)
```

Queries for MASS and FODOR databases after translation into SQL:

```
SELECT * FROM MASSInfo

SELECT * FROM FODORInfo
```

Figure 11: Processing a simple query using the example articulation axioms

SQL query requesting the phone numbers of all hotels with restaurants:

```
SELECT name, phoneNumber FROM Lodging
       WHERE Lodging.hasAmenities = "Restaurant"
```

Query after translation to CycL:

```
(and (allInstanceOf LODGING Lodging)
     (allInstanceOf R Restaurant)
     (hasAmenities LODGING R)
     (name LODGING NAME)
     (phoneNumber LODGING NUMBER))
```

Queries (in CycL) for MASS and FODOR databases after translation using articulation axioms:

```
(and (allInstanceOf LODGING MASSInfo)
     (name LODGING NAME)
     (allinstanceOf AR AmenityRelationship)
     (inAmenityRelationship LODGING AR)
     (involvesAmenities AR AMENITY)
     (allInstanceOf AMENITY AmenityInfo)
     (amenityCode AMENITY 4)
     (phone LODGING NUMBER))

(and (allInstanceOf LODGING FODORInfo)
     (name LODGING NAME)
     (facilities LODGING F)
     (facilityCode F Restaurant)
     (phones LODGING PHONE)
     (phoneNum PHONE NUMBER))
```

Queries for MASS and FODOR databases after translation into SQL:

```
SELECT name, phone FROM MASSInfo, AmenityRelationship, AmenityInfo
       WHERE MASSInfo.name = AmenityRelationship.name
         AND AmenityRelationship.amenityCode = AmenityInfo.amenityCode
         AND AmenityInfo.amenityCode = 4

SELECT name, phoneNum FROM FODORInfo, FODORPhone, FODORFacility
       WHERE FODORInfo.facilities.facilityCode = "Restaurant"
```

Figure 12: Processing a more complex query using the example articulation axioms

Axiom 1: In the MASS database, the entity MASSInfo represents lodging:

```
(equivalence
  (isTrue GC (allInstanceOf LODGING Lodging))
  (isTrue ERMASSMicrotheory (allInstanceOf LODGING MASSInfo)))
```

Axiom 2: In the MASS database, hotels have telephones:

```
(equivalence
  (isTrue GC
    (and (allInstanceOf LODGING Lodging)
         (phoneNumber LODGING NUMBER)))
  (isTrue ERMASSMicrotheory
    (phone LODGING NUMBER)))
```

Axiom 3: In the MASS database, a hotel with `amenity-code=4` has a restaurant on its premises:

```
(equivalence
  (isTrue GC
    (and (allInstanceOf LODGING Lodging)
         (allInstanceOf R Restaurant)
         (hasAmenities LODGING R)))
  (isTrue ERMASSMicrotheory
    (and (allInstanceOf LODGING MASSInfo)
         (allinstanceOf AR AmenityRelationship)
         (inAmenityRelationship LODGING AR)
         (involvesAmenities AR AMENITY)
         (allInstanceOf AMENITY AmenityInfo)
         (amenityCode AMENITY 4))))
```

Figure 13: Articulation axioms relating the MASS database schema to the Cyc global schema

Axiom 4: In the FODOR database, the entity FODORInfo represents lodging:

```
(equivalence
  (isTrue GC (allInstanceOf LODGING Lodging))
  (isTrue OOFODORMicrotheory (allInstanceOf LODGING FODORInfo)))
```

Axiom 5: In the FODOR database, hotels may have restaurant facilities:

```
(equivalence
  (isTrue GC
    (and (allInstanceOf LODGING Lodging)
         (allInstanceOf R Restaurant)
         (hasAmenities LODGING R)))
  (isTrue OOFODORMicrotheory
    (and (allInstanceOf LODGING FODORInfo)
         (facilities LODGING F)
         (facilityCode F Restaurant))))
```

Axiom 6: In the FODOR database, the hotel class can have a set of instances from the telephone class:

```
(equivalence
  (isTrue GC
    (and (allInstanceOf LODGING Lodging)
         (phoneNumber LODGING NUMBER)))
  (isTrue OOFODORMicrotheory
    (and (allInstanceOf LODGING FODORInfo)
         (phones LODGING PHONE)
         (phoneNum PHONE NUMBER))))
```

Figure 14: Articulation axioms relating the FODOR database schema to the Cyc global schema

After a schema is integrated, the resultant articulation axioms enable the translation of transactions from a local schema context to a more general context GC (the global schema), and vice versa.

7 Discussion

We describe in this paper an ongoing experiment in resource integration that we are conducting using the large knowledge base Cyc. The resources we consider are database systems. We present the basic Cyc units that have been defined to represent different types of schemas. Integration of these schemas is based on the definition of articulation axioms between two contexts: the context of the preintegrated schema and a global schema context provided by the Cyc ontology. Our 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 need only $2N$ mapping functions for N resources that we wish to integrate in the 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 is propagated only to the

global schema, and only one mapping function has to be recalculated. We believe that this calculation would typically involve only a few rules of the mapping.

Semantic Integrity and Integration Support Tool

In accordance with the above principles and methodology for schema integration, we are developing a a Semantic Integrity and Integration Support (SIIS) tool as a part of the semantic services of Carnot [Cannata 1991]. The SIIS automates and enables the implementation and subsequent use of integrated information resources. It includes the preintegration tool (Section 4.2) and provides capabilities for 1) establishing a global schema that captures the explicit semantics represented in each of the component resources and the implicit semantics perceived by users, 2) managing such a global schema and its associated language, and 3) processing update and query transactions.

As we show in Section 5, the integration process cannot be performed automatically using current technology [Sheth and Gala 1989]. The SIIS tool *interactively* assists a designer in building a global schema and the requisite mapping rules. Figure 9 illustrates how this occurs. Further, the tool provides functionalities to execute multiresource queries and updates, as illustrated in Figure 10.

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, 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 are developing 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 global schema consists of all of the contexts representing the different domains of the integrated databases plus the base context of Cyc. Because the global schema is significantly larger than the strict set of units involved in integrating the databases, a user can issue more general queries than if a merged schema were used. However, its large size also makes it difficult for the user to issue queries. There is a need for a definition of a minimal global schema including only the units gotten from the integration of local schemas. Using the fact that every local schema belongs to a particular context, it should be easy to determine the minimal global schema and build an external representation from it that can be used for querying and updating.

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.

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.

[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.

[Cannata 1991] Philip E. Cannata, "The Irresistible Move towards Interoperable Database Systems," *First International Workshop on Interoperability in Multidatabase Systems*, Kyoto, Japan, April 7-9, 1991.

[Collet and Huhns 1991] Christine Collet and Michael N. Huhns, "Semantic Integrity and Integration Support in Carnot," MCC Technical Report Number ACT-OODS-073-91(Q), Microelectronics and Computer Technology Corporation, Austin, TX, March 1991.

[Elmagarmid *et al.* 1990] A. K. Elmagarmid, Y. Leu, W. Litwin, and M. Rusinkiewicz, "A Multidatabase Transaction Model for InterBase," *Proceedings of the 16th VLDB Conference*, Brisbane, Australia, February 1990, pp. 507–518.

[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, July 1985, pp. 253–278.

[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.

[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 1990] 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. Krishnaswamy, A. Savasere, and W.K. Whang, "Federated Architecture for Heterogeneous Information Systems," *NSF Workshop on Heterogeneous Databases*, December 1989.

[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.

[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.

[Veijainen 1990] J. Veijainen, *Transaction Concepts in Autonomous Database Environments*, R. Oldenbourg Verlag, Germany, 1990.

[Wang and Madnick 1989] Richard Wang and Stuart E. Madnick, "Facilitating Connectivity in Composite Information Systems," *Data Base*, Fall 1989, pp. 38–46.

[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. The three 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 notation of Orion [Kim *et al.* 1989], MCC's object-oriented database management system):

```
FODORInfo(name          : String,
          address       : FODORAddress,
          comment       : String,
          location      : 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)`

The object classes are depicted in Figure 15.

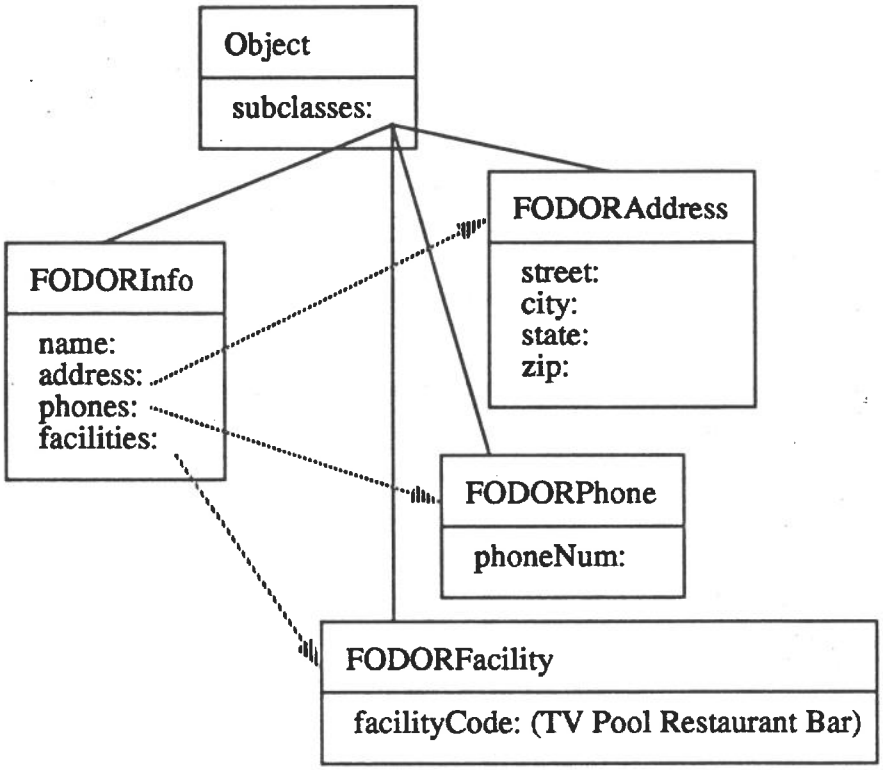


Figure 15: Object classes for the FODOR database

- 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

[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.

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

The Entity **AmenityInfo** with the attribute

```
amenityCode*
```

The relationship **MASSAmenity** relates the entities **AmenityInfo** and **MASSInfo**, and has no attributes. The cardinality of the relationship is $m : n$, i.e., one entity of **MASSInfo** may be associated with m, $m > 0$, entities of **AmenityInfo**, and one entity of **AmenityInfo** may be associated with n entities of **MASSInfo**. Further, there is no dependency constraint among the entities in the relationship, i.e., the existence of an entity of **MASSInfo** does not depend on an entity of **AmenityInfo**. The primary key of **MASSAmenity** is composed of the primary keys of the participating entities, i.e., **name** and **amenityCode**. An entity-relationship diagram for this database is shown in Figure 16.

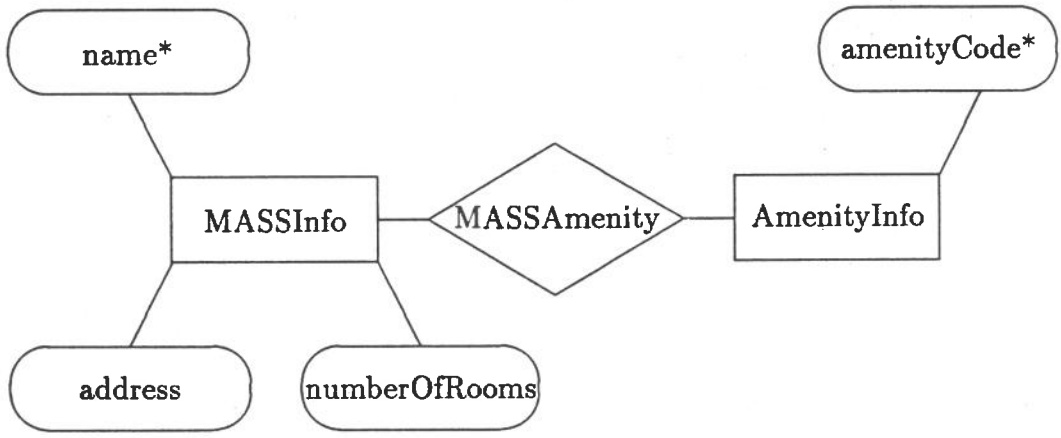


Figure 16: Entity-relationship diagram for the MASS database (only a few of the attributes are shown)