A Tool for Integrating Conceptual Schemas and User Views

Amit P. Sheth* and James A. Larson
Honeywell Corporate Systems Development Division
1000 Boone Ave North
Golden Valley MN 55427
sheth@rdcs.sm.unisys.com

Aloysius Cornelio and Shamkant B. Navathe
Database Research and Development Center
University of Florida Gainsville FL 32611
sham@ufl.edu.csnet

ABSTRACT

Schema integration is important in two contexts, logical database design (in centralized DBMS) and global schema design (in distributed DBMS). Performing an integration on real-life schemas without a tool can be very difficult, tedious and error prone. We have developed an interactive tool to assist database designers and administrators (DDA) in integrating schemas. It collects the information required for integration from a DDA, performs essential bookkeeping, and integrates schemas according to the semantics provided. This paper presents the capabilities of this tool by discussing the integration methodology and the user interface of the tool.

Keywords: *schema integration, view integration, database design and integration, conceptual data model, ECR model, interactive tool, database designer and administrator*

1. Introduction

With the diversity of data models and database management systems, the development of schema integration tools is becoming increasingly important. Schema integration arises in two different contexts:

(a) Logical database design. Several views are merged to form a logical schema describing the entire database. User queries and transactions specified against each view are mapped to the logical schema.

(b) Global schema design. Several databases already exist and are in use. The objective is to design a single global schema that represents the contents of all these databases. This global schema can then be used as an interface to the diverse databases. User queries and transactions specified against the global schema are mapped to the schemas supported by the relevant databases.

In both contexts it is necessary to integrate two or more schemas, which we call *component schemas*, into a single integrated schema. Database designers and administrators (DDAs) are responsible for integrating schemas. We have designed and developed a schema integration tool [Cornelio and Sheth 86] that permits them integrate schemas interactively. In the present paper, we describe the capabilities and some implementation details of our tool. The methodology is described in detail in [Elmasri et al 86, Navathe et al 84, Elmasri and Navathe 84]. It consists of four phases (see Figure 1):

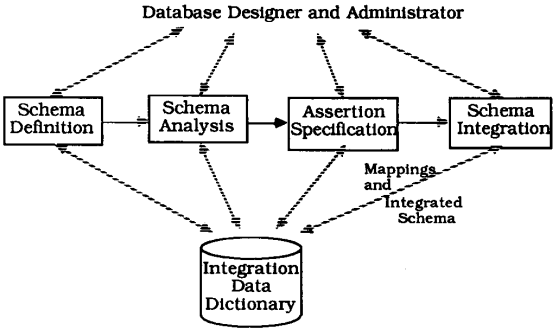


Figure 1: Four phases of schema integration

Phase 1: Schema collection:

First, component schemas are defined or identified. Before integration, all component schemas must be specified using a common data model. We use a variation of the Entity-Relationship (ER) model called the Entity-Category-Relationship (ECR) model and its data description language as the common data model (ECR model is described briefly in section 2). If a component schema is defined in a data model other than ECR model, it must be translated to the ECR model. Navathe and Awong [Navathe and Awong 87] have developed a detailed procedure for interrogating a DDA to extract the semantics about relational and hierarchical database schemas and then to map them automatically in ECR model. A few other tools incorporate this schema translation step to some extent (e.g.,[Reiner et al 84]).

Phase 2: Schema analysis:

In some cases, schema constructs in one component schema may need to be changed to become more compatible with equivalent schema contructs in other component schemas. For example, an attribute in one component schema may correspond to an entity type in another. One of the two representations must be chosen so that equivalent concepts can be integrated. Other incompatibilities that may need to be considered during schema analysis are differences in naming conventions, scales/units, domain constraints, and other factors.

Schema analysis leads to the identification of *attribute equivalence* classes. The DDA here recognizes the equivalence among attributes of object classes (an object is an entity set or categories) and between relationship sets belonging to the schemas to be integrated. This is governed by the real world meaning of these objects. The equivalence information forms the basis for choosing the pairs of object classes that can be integrated.

* Present Address: UNISYS Corp., 2400 Colorado Avenue, Santa Monica CA 90406. (213) 829-7511/Ext. 5330.

Phase 3: Assertion specification:

Next, correspondences among the object classes and relationship sets across the different component schemas are specified as assertions by the DDAs. The types of assertions that may be specified between two object classes are discussed in Section 2. Specification of assertions is the main vehicle for providing the semantics of integration. Following this, the assertions must be checked for consistency so that no group of assertions leads to a contradiction. Once the assertions are collected, and no contradictions exist among them, integration can proceed.

Phase 4: Integration:

In this phase, the object classes and relationship sets of two schemas are integrated. During object class integration, object classes may be (a) merged into a single object class, (b) placed into an IS-A lattice with some object classes being generalizations of other object classes, or (c) kept as separate object classes. Following the integration of object classes, relationship sets are integrated. Relationship set integration can be performed in a manner similar to object class integration. The lattices resulting from integration of object classes and relationship sets are merged to form the integrated schema.

Following integration, mappings between each component schema and the integrated schema are generated. Mappings are used to translate requests in an operational system after integration, and these mappings differ in the two integration contexts. In the logical database design context discussed at the beginning of the section, the mappings are used to convert requests against component schemas (views) into requests against the integrated schema. In the global schema design context, the mappings are used to map requests against the integrated schema (global schema) into requests against the component schemas.

These phases are based on an extension of several schema integration methodologies, including ours [Navathe and Gadgil 82]. Several methodologies for schema integration [Batini and Lenzerini 83, Dayal and Hwang 84, Mannino and Effelsberg 84, Teorey and Fry 82, Wiederhold and Elmasri 79, Yao et al 82] are compared in a survey paper by [Batini et al 86]. As classified in this survey, our methodology is unique in that it performs *n-ary* integration; however, it is interactive in nature since the assertions are collected by doing pair wise comparisons of attributes, objects, and relationships. A number of systems for computer aided database design have been implemented; for example, INCOD [Atzeni 82] [Ceri 83], IMT [Lundberg 82] and DDEW [Reiner et al 84]. Like these systems, our system performs many bookkeeping details to keep track of objects in the schema being designed. Our system also provides a tool for schema integration. This paper summarizes this tool. Briefly, our schema integration tool uses a heuristic that ranks pairs of objects which might be integrated. The DDA then reviews this ranked sequence of object pairs and specifies the exact relationship between some of the pairs. Any inconsistencies are detected. (For example, if Employee is equivalent to Person, and Person is equivalent to Worker, then Worker cannot be a subset of Employee.) By performing the transitive closure of existing relationships between pairs of objects, the relationships between additional pairs of objects can be determined automatically. (For example, in another context, if Worker is subset of Employee and Employee is subset of Person, then Worker must be subset of Person.)

In this section, we summarized the methodology we use for schema integration. The remainder of this paper is organized as follows. In section 2 we discuss the basic concepts of the ECR data model and the object class integration. Section 3, the main section of the paper, discusses various aspects of the tool with emphasis on how the user interface helps the DDA to interactively specify the information required for schema integration. For the sake of brevity, the basic algorithms and processing steps are not discussed in detail. In Section 4, we discuss possible future extensions of this tool.

2. Basic Concepts

The data model used in our work is the Entity-Category-Relationship (E-C-R) model [Elmasri et al 85], which is an extension of the popular Entity-Relationship (E-R) model proposed by [Chen 76]. These models and several others, which are sometimes classified as conceptual data models, tend to describe the semantics of data by classifying the application domain into entities (which may be objects, events, things, concepts) and relationships among the entities. We chose this model because it is easy for a DDA to understand and it allows us to represent semantics required for integration. The E-C-R model includes the following extensions to the E-R model:

(1) The concept of a category is introduced to represent generalization hierarchies and subclasses.

(2) Structural constraints on relationships are used to specify how entities may participate in relationships.

The E-C-R model, like the E-R model, views the world as consisting of entities and relationships among entities. Entities and relationships have attributes that provide information. Entities with similar basic attributes are classified into entity sets. Entity sets are disjoint in that a given entity can be a member of only one entity set.

A category is a subset of entities from an entity set. Categories allow the representation of generalization hierarchies. A category inherits the attributes of the object class (entity set or category) over which it is defined.

A relationship associates entities from two or more object classes. A collection of relationships of the same type involving the same object classes is called a relationship set. For semantic integrity, rules govern the possible ways in which entities of an object class can participate in a relationship. We call these rules the structural constraints of the relationship. We can specify a type of structural constraint, called a cardinality constraint, on the participation of an object class in a relationship by using two numbers (i_1, i_2), where $0 \leq i_1 \leq i_2$ and $i_2 > 0$. The numbers mean that each entity that is a member of the object class must participate in at least i1, and in at most i2 relationship instances in the relationship set.

At the center of schema integration is the concept of **object integration**, which is based on domains of the object classes. The domain of an object class is the set of object instances in that class at a given time. Two classes from different views may have the same, different, or overlapping domains. The domain of an object class is the set of real-world object instances that are modeled by the object class. We have identified several relationships which may hold between two object classes from different schemas [Elmasri and Navathe 84, Navathe et al 86]. An *assertion* means specification of a relationship between domains of two object classes in different schema. The relationships between object domains and the assertions can be as follows:

(1) The object classes have identical domains ("equals" assertion). In this case, the object classes may be combined into a single object class. For example (Figure 2a), the Department entity set from schema sc1 and the Department entity set from schema sc2 are determined to have identical domains, and are integrated into a single equivalent entity set called E_Department.

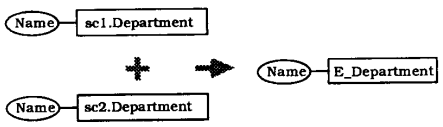


Figure 2a: Identical Domains

(2) The domain of one object class is contained in the domain of another object class ("contains" assertion). In this case, the contained object class becomes a category of the containing object class. For example (Figure 2b), the domain of the Student entity set contains the domain of the Grad_student entity set. After integration, Grad_Student becomes a category of the Student object class.

(3) The domains of two object classes overlap, but neither is contained in the other ("may be" assertion). In this case, a new derived entity class is generated that contains the union of the original domains, and the original entity sets become categories of the new entity set. For example (Figure 2c), The domain of Grad_student and the domain of Instructor overlap, but neither is a subset of the other. After integration, a new derived entity set called D_Grad_Inst is created, with Grad_student and Instructor as categories.

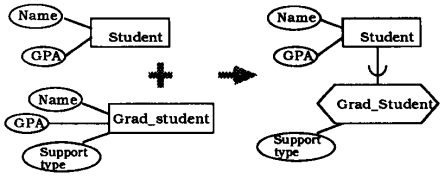


Figure 2b: Contained Domains

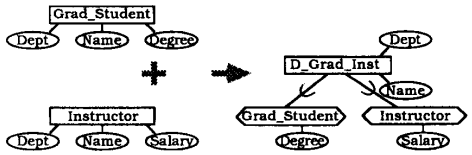


Figure 2c: Overlapping Domains

(4) The domains of two objects are disjoint, but can still be integrated ("disjoint integrable" assertion). In this case, a new derived entity set is generated containing the union of the original domains, and the original entity sets become categories of the new entity set. Figure 2d illustrates the Secretary and Engineer entity sets, which have disjoint domains but can be integrated together as the derived entity set D_Secr_Engi which represents the concept of employee. The DDA must make the subjective decision to integrate or not based on the anticipated utility of the new object class.

(5) The domains of two objects are disjoint and are not integrated ("disjoint nonintegrable" assertion). Figure 2e illustrates the Under_Grad_Student entity set and the Full_Professor entity set. These entity sets have disjoint domains. Because no useful purpose exists for a new entity set consisting of undergraduate students and full professors, the DDA would likely determine that these two disjoint objects should not be integrated.

We use the term "assertion" to mean a specification of a relationship between the domains of two object classes in different schemas. Only by understanding the semantics of the application can the DDAs determine assertions. In the example in item (5) above, the DDA may think that the concept of "person" is useful, in which he may specify a "disjoint integrable" assertion. To make assertion specification easier, we use a heuristic to order pairs of objects for review by the DDA. This heuristic is based on a *resemblance function* which measures the ratio of similar attributes to total attributes of two objects.

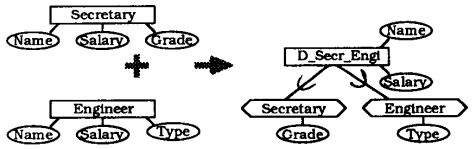


Figure 2d: Disjoint Domains

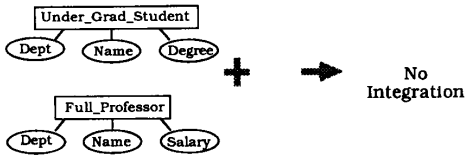


Figure 2e: Non-integrable Domains

To apply this resemblance function, we need a way to determine if two attributes (of different objects) are equivalent. We use the term "equivalence" to mean a specification of the similarity of the domain of two attributes. [Larson et al 87] present a theory of attribute equivalence. Here we will use a simplification of that theory in which two attributes are determined to be either equivalent or nonequivalent. Our schema integration tool helps the DDA determine and keep track of equivalent attributes.

3. Description of the Tool

In this section, we first give an overview of the tool and then discuss the four phases used during schema integration in the following subsections.

3.1. Overview

Our schema integration tool is written in C and runs on Apollo in the UNIX environment. The tool is interactive; the user interface of the tool is menu and form based and largely terminal independent. All screen and cursor movements are performed using a UNIX library package called *curses*. Each screen is made up of multiple windows, some of which can be scrolled to supply and display additional information.

When the tool is invoked, the user is presented with the main menu, which describes the tasks required for integration. The main menu is shown in Screen 1.

```
SCHEMA INTEGRATION PACKAGE

MAIN MENU
***Integrate Entities and Categories before Relationships***

1. Schema Information Collection

2. Equivalence Class Creation and Deletion (entities and categories)

3. Equivalence Class Creation and Deletion (Relationships)

4. User Assertions (entities and categories)

5. User Assertions (Relationships)

6. Display Integrated Schema

0. Exit

Enter Selection =>
```

Screen 1. Main menu of the tool

The six tasks in the main menu closely follow the four phases of schema integration methodology described in the first section. The first task corresponds to the schema collection phase in which the DDA interactively defines the schemas to be integrated. The second and third phases are carried out in two tasks each, one for object classes and one for relationship sets. The second and third tasks correspond to the second phase. Our tool does not provide an automated aid for schema modification. The DDA manually resolves such conflicts and changes the schema by going back to the first phase. The tool allows interactive specification of attribute equivalences. The third phase is carried out in tasks four and five. Finally, the fourth phase is carried out in task six, where the DDA can view the results of integration. The DDA generally performs the tasks in the serial order.

3.2. Schema Collection

This is the first phase in which the DDA defines the schemas (schemata) to be integrated. A user can define any number of schemas, but only two schemas can be integrated at a time. A result of integration of two schemas can be integrated with another schema; thus multiple schemas can be integrated. For ease of explanation, we will use examples corresponding to a possible integration of two simple schemas, schema sc1 shown in Figure 3 and schema sc2 shown in Figure 4.

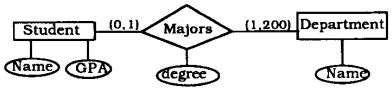


Figure 3: Input Schema sc1

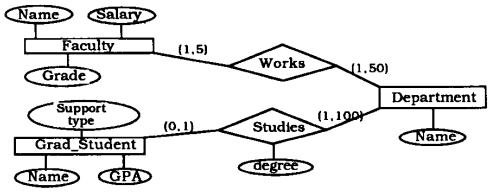


Figure 4: Input Schema sc2

The Schema Collection phase consists of a set of five screens presented to the DDA to define the input schemas. The first screen is the Schema Name Collection Screen (see Screen 2) in which the user defines the names of the schemas to be integrated. Menu options are selected to add or delete information corresponding to the last schema name specified, to specify the name of previously defined schema to be updated, or to exit.

Upon selecting the add or delete operation, the Structure Information Collection Screen (see Screen 3) is displayed. Here the user enters information pertaining to all the structures (of type entity sets, category, or relationship sets) belonging to the schema under consideration. For a structure of type category, the Category Information Collection Screen is used to specify all the entities and categories connected to a category. For a structure of type relationship, information about the entities it connects is collected using the Relationship Information Collection Screen (see Screen 4). For all objects, the Attribute Collection Screen (see Screen 5) is presented to collect information about each of the attributes of the object.

SCHEMA COLLECTION
<Schema Name Collection Screen>

Schema Name

1> sc1
2> sc2

Choose: (A)dd (D)elete (U)pdate (E)xit :

Screen 2. Schema Name Collection Screen

SCHEMA COLLECTION
<Structure Information Collection Screen>
SCHEMA NAME: sc1

	object Name	Type(E/C/R)	# of attributes
1>	Student	e	2
2>	Department	e	1
3>	Majors	r	1

Choose: (S)croll (A)dd (D)elete (U)pdate (E)xit :

Screen 3. Structure Information Collection Screen

SCHEMA COLLECTION
<Relationship Information Collection Screen>

SCHEMA NAME: sc1 RELATIONSHIP NAME: Majors

Object Name	Type(E/C)	CARDmin	CARDmax	Role
Student	e	0	1	gets
Department	e	5	n	in

Choose: (A)dd (D)elete (E)xit :

Screen 4. Relationship Information Collection Screen.

SCHEMA COLLECTION
<Attribute Information Collection Screen>

SCHEMA NAME: sc1 OBJECT NAME: Student TYPE: e

	Attribute Name	Domain	Key (y/n)
1>	Name	char	y
2>	GPA	real	n

Choose: (S)croll (A)dd (D)elete (E)xit :

Screen 5. Attribute Information Collection Screen

3.3. Equivalence Class Specification

When two schemas are being integrated, potential similarity exists among every pair of structures when (1) both structures are either entity sets or categories or both structures are relationships, and (2) each structure belongs to different schema. The tool enables a DDA to deal with the large number of pairs by interactively leading the DDA to specify equivalences among pairs of attributes belonging to different objects. Two attributes are determined to be equivalent based on several characteristics, including uniqueness (the "key" property), cardinality, domain, etc. (see [Larson et al 87] for a detailed discussion.)

The attribute equivalences are the basis of a resemblance function used to to generate an ordered list of object pairs using the following simple heuristics. The likelihood of two objects being

integrated is proportional to the percentage of equivalent attributes. In other words, object classes with many equivalent attributes could represent the same concept and thus become candidates for integration. The higher the percentage of equivalent attributes between two objects, the more likely they are to be integrated with stronger assertions. Besides forming the basis for the resemblance function, attribute equivalence is also useful in generating mappings during schema integration.

This phase has two subphases: defining equivalences among the attributes of object classes (by choosing option 2 of the main menu) and defining equivalences among the attributes of relationship sets (by choosing option 4 of the main menu). We will discuss only the first subphase because the second subphase is similar.

After this phase is chosen by selecting item 2 on the main menu, the tool uses three screens to collect the attribute equivalence information. First, the DDA is presented with the Schema Name Selection Screen, where he/she chooses two schemas that are being integrated (this phase can be successively invoked for integrating other pairs of schemas). Next, the DDA is presented with the Entity/Category Name Selection Screen (see Screen 6), which shows all the entity types and categories of the two schemas being integrated. Here the DDA picks one object class (entity or category) from each schema whose attributes may be equivalent.

Finally, the DDA is presented with the Equivalence Class Creation and Deletion Screen (see Screen 7), where the DDA identifies attributes in the same equivalence class. An equivalence class consists of all the attributes defined to be equivalent by the DDA. For example, the DDA determines that sc1.Student.Name is equivalent to sc2.Grad_student.Name. The tool then changes the value of Eq_Class # of one to that of the other. At the end of this phase, there may be (depending on the equivalences defined by the DDA) an equivalence class consisting of sc1.Student.Name, sc2.Faculty.Name and sc2.Grad_student.Name.

The tool maintains a structure called Attribute Class Similarity (ACS) matrix, which maintains all the equivalence class definitions given in this phase. Upon exiting this phase, the tool derives an Object Class Similarity (OCS) matrix from the ACS matrix, where each element of the matrix specifies the number of equivalent attributes between two objects specified by the row and column order. OCS matrix contains information to generate an ordered list of object class pairs corresponding to their likelihood of being integrable with stronger assertions. The ordering algorithm is described in more detail in [Elmasri et al 86].

EQUIVALENCE CLASS SPECIFICATION
<Entity/Category Name Selection Screen>

<sc1>	<sc2>
1> Student	1> Grad_student
2> Department	2> Faculty
	3> Department

(S)croll (C)hoose objects (L)ist attributes (E)xit =>

Screen 6. Entity/Category Name Selection screen.

EQUIVALENCE CLASS SPECIFICATION
<Equivalence Class Creation and Deletion Screen>

(schema.object1) sc1.Student		(schema.object2) sc2.Grad_student	
Attribute Name	Eq_class #	Attribute Name	Eq_class #
1> Name	1	1> Name	1
2> GPA	2	2> GPA	6
		3> Support_type	7

(S)croll (A)dd or (D)elete from equiv. class (E)xit =>

Screen 7. Equivalence class creation and deletion screen.

3.4. Assertion Specification

In this phase, the DDA specifies one of the five assertions for each ordered pair of object classes and relationship sets. The phase also consists of two subphases, one for entity and category object classes (invoked by selecting option 3 from the main menu) and one for relationship sets (invoked by selecting option 5 from the main menu). For the sake of brevity, we will discuss only the first subphase.

In this phase, the tool interacts with the DDA with two screens. The first is the Assertion Collection For Object Pairs (see Screen 8), which presents ordered object pairs and an attribute ratio for each pair that specifies (# of equivalent attributes)/(# of equivalent attributes + # of attributes in the smaller object class). Thus a value of 0.5 for attribute ratio specifies that every attribute in one object class has an equivalent attribute in the other object class. The DDA is then asked to specify an assertion the for each pair. The tool also allows the DDA to review and modify any assertion when he completes giving assertions.

ASSERTION SPECIFICATION
<Assertion Collection For Object Pairs Screen>

Schema_Name1.Obj_Class1	Schema_Name2.Obj_Class2	ATTRIBUTE RATIO	ENTER ASSERTION
sc1.Department	sc2.Department	0.5000	=>1
sc1.Student	sc2.Grad_student	0.5000	=>3
sc1.Student	sc2.Faculty	0.3333	=>4

ENTER:1 - OB_CL_name_1 equals OB_CL_name_2
2 - OB_CL_name_1 'contained in' OB_CL_name_2
3 - OB_CL_name_1 'contains' OB_CL_name_2
4 - OB_CL_name_1 and OB_CL_name_2 are disjoint but integratable
5 - OB_CL_name_1 and OB_CL_name_2 may be integratable
0 - OB_CL_name_1 and OB_CL_name_2 are disjoint & non-integratable

Screen 8. Assertion collection for object pairs from sc1 and sc2

Assertions between every pair of object classes are stored in an Entity Assertion matrix, where element (i,j) in the matrix represents the assertion between object classes i and j. Some of the assertions may be specified by the user; the rest may be derived using rules of transitive composition of assertions (such as if a $\subseteq$ b and b $\subseteq$ c then a $\subseteq$ c). At the same time assertions are derived, the tool also checks for consistency of a newly defined or derived assertion with the previously defined or derived assertion. If a conflict is found, the tool uses the Assertion Conflict Resolution Screen (see Screen 9). This screen tells the DDA of the conflicting assertions. If an assertion is derived, the screen also specifies all the relevant assertions used in the derivation. Then the DDA is asked to change the assertions so that they do not conflict. For example, Screen 9 shows conflicts among object classes of schemas sc3 and sc4 (not shown). The first line shows the assertion derived from the earlier specified assertions in lines 3 and 4. The second line shows the newly specified assertion which conflicts the assertion in line 1. To resolve the conflict the DDA may change earlier assertion in line 3 (possibly to a "0" or "5") realizing that all instructors are not grad_students.

Specifying assertions requires interacting with the DDA and cannot be completely automated. This is because two schemas can be integrated in different ways depending on the intended semantics of their use. Details of the assertion specification and conflict deletion algorithms are found in [Elmasri et al 86].

ASSERTION SPECIFICATION
<Assertion Conflict Resolution Screen>

SCHEMA_NAME1.OBJ_CLASS1	SCHEMA_NAME2.OBJ_CLASS2	CURRENT ASSERTION	NEW ASSERTION
sc3.Instructor	sc4.Student	2	<derived>(CONFLICT)
sc3.Instructor	sc4.Student	0	<new>(CONFLICT)
sc3.Instructor	sc4.Grad_student	2	
sc4.Grad_student	sc4.Student	2	

ENTER: 1 - OB_CL_name_1 equals OB_CL_name_2
2 - OB_CL_name_1 'contained in' OB_CL_name_2
3 - OB_CL_name_1 'contains' OB_CL_name_2
4 - OB_CL_name_1 and OB_CL_name_2 are disjoint but integratable
5 - OB_CL_name_1 and OB_CL_name_2 may be integratable
0 - OB_CL_name_1 and OB_CL_name_2 are disjoint & non-integratable

Screen 9. Assertion Conflict Resolution Screen.

3.5. Integration

Upon completing the third phase, the tool performs integration. This involves creating clusters of entity sets. A cluster is a group of related objects that are connected by any assertion except disjoint disintegrable. The concept of cluster helps in partitioning the schemas to more manageable subsets. Using the concepts of object integration discussed before, these objects are then integrated. First, entity sets and categories are integrated to form a lattice structure of interdependent object classes. Next, relationship sets are integrated to form lattices of relationship sets. Finally, two lattices are merged to form the integrated schema (see [Elmasri et al 86] for more details). The result of integrating schema sc1 of Figure 3 and schema sc2 of Figure 4 for one likely set of assertions is shown in Figure 5.

The result of schema integration can be viewed using the set of eight screens arranged in a hierarchy, each representing different details of the integrated schema. Figure 6 shows control flow of the screens in this phase, where the annotation on an arc between two screen shows the menu choice made in the screen at the tail of the arc to invoke the screen at the head. The lables on the arcs show that choice to be made at the screen on the tail of the arc to invoke the screen at the head of the arc.

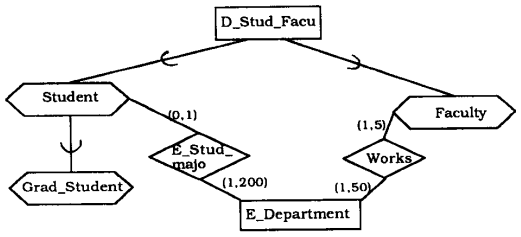


Figure 5: Result of integrating sc1 and sc2 (attributes not shown)

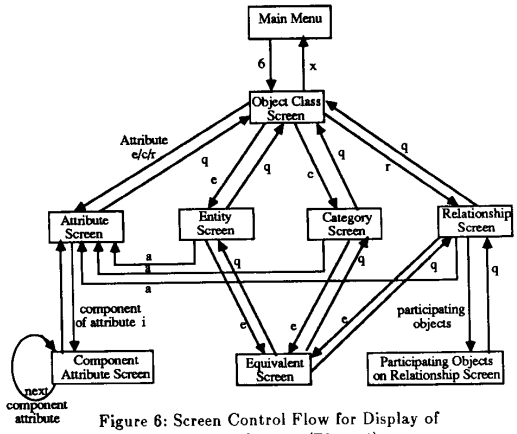


Figure 6: Screen Control Flow for Display of Integrated Schema (Phase 4)

The main screen is the Object Class Screen (see Screen 10), which shows all the object classes and relationship sets in the integrated schema. The number in the parenthesis following "entities", "categories", and "relationships" shows the length of the corresponding list and indicates if scrolling is required. An object class (relationship set) with a "D_" prefix is a derived object class (relationship set), resulting from integrating object classes (relationship sets) connected with "may be", "contains", "contained in", or "disjoint integrable" assertion. Similarly, an attribute with a "D_" prefix is a derived attribute generated during the integration. An object class (relationship set) with an "E_" prefix is an equivalent object class (relationship set) resulting from integrating two object classes (relationship sets) with the "equals" assertion. Four screens can be invoked from the Object Class Screen to obtain more details about the integrated schema. The Entity Screen, Category Screen or Relationship Screen appears when the corresponding choice is made in the bottom menu. An Entity Screen of an entity shows all the children object classes of that entity. A Category Screen (see Screen 11) shows all the parent and children object classes of the category. For example, in our integrated schema, for object class Student, the parent object class is D_Stud_Facu and the child object class is sc2.Grad_stud. A Relationship Screen is similar to a Category Screen. The fourth screen is the Attribute Screen, which is used to view all the attributes of any object class. For a derived attribute, a Component Attribute Screen (see Screens 12a and 12b) is invoked to view each attribute in the original schemas from which the attribute in the integrated schema is derived. For example, attribute Name of object class Student in the integrated schema has two component attributes-- sc1.Student.Name and sc1.Grad_student.Name. An Equivalent Screen can be invoked from an Entity Screen, Category Screen or Relationship Screen to view the objects from which an equivalent object class is obtained. Finally, a Participating Objects In Relationship Screen is invoked from a Relationship Screen to see the entities and categories tied to a relationship.

INTEGRATED SCHEMA
<Object Class Screen>

Entities(2)	Categories(3)	Relationships(2)
E_Department	Student	E_Stud_Majo
D_Stud_Facu	Grad_student	Works
	Faculty	

Do you want to examine: <m>ore object class names,
<a>ttributes, <c>ategories, <e>ntities,
<r>elationships, <x> to exit =>

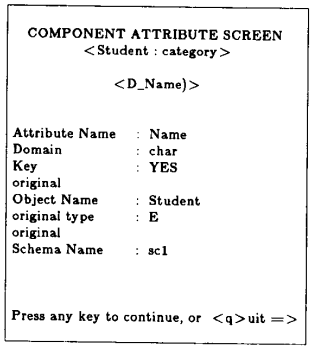
Screen 10. Object Class Screen

INTEGRATED SCHEMA
<Category Screen>
<Student>

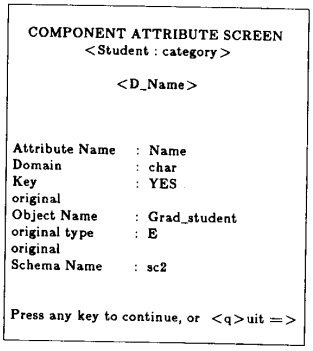
Parent Object(1) (type)	Child Object(1) (type)
1) D_Stud_Facu (e)	sc2.Grad_student (c)

Do you want to examine:
<e>quivalent object classes
<a>ttribute info, or <q>uit =>

Screen 11. Category Screen for Student



Screen 12a. Component Attribute Screen
(first component attribute of D_Name)



Screen 12b. Component Attribute Screen
(second component attribute of D_Name)

4. Future Work and Discussion

While our tool performs the basic functions of schema integration, the following enhancements can increase its utility.

Enhancements are most needed in the schema analysis phase and fall into two categories. "Syntactic processing enhancements" include the introduction of string matching heuristics to identify potentially equivalent attributes. A dictionary of synonyms and antonyms would also be useful in detecting candidate pairs of equivalent attributes. SIS [de Souza 86] describes several resemblance functions (such as "to have similar names" or "to have identifiers with similar names"). Using a weighted sum of products of several resemblance functions, pairs of objects can be sorted according to their mutual resemblance. Our system would benefit from having additional resemblance functions. The resemblance function among objects could be possibly extended to derive a resemblance function among schemas which could be particularly useful in picking similar schemas for integration in a binary approach.

"Semantic processing enhancements" include heuristics to identify corresponding objects of different constructs. For example, in one schema, a marriage between two people may be represented as an entity set, while in another schema a marriage may be represented as a relationship between the entity sets Male and Female. One approach to this problem [Larson et al 87] is to two different types of constructs if they have several common attributes. For example, the entity set *marriage* and and the relationship set *marriage* could be identified as equivalent if they both have attributes *marriage-data, marriage-location, number of children*, etc. We feel that in many cases, common attributes indicate that constructs of different types may have corresponding roles in the different schemas and hence are subjects for integration. Much work remains to be done in this area.

Our tool could also benefit from several practical enhancements. For example, it would be more useful if it were integrated into a distributed database design environment along with other database design tools. The algorithms proposed by [Navathe and Awong 87] may be implemented into a tool to translation schemas from conventional data models to ECR. The output of a schema translation tool could go directly into our tool, with the result feeding into a physical database design tool. A common representation of the database objects and the mappings between them could be kept in a data dictionary available to all of the tools. Finally,a graphical interface for displaying and browsing schemas [Larson 86] would make the system even more easy to use.

Acknowledgements: We thank Prof. Ramez Elmasri for his contributions to the methodology and algorithms upon which this tool is based and A. Fuller, P. Ho, L. Munsun and Ms. X. Zhu for their help in implementing the tool.

REFERENCES

[Atzeni et al 82] Atzeni, et al. "A Computer Aided Tool for Conceptual Data Base Design," in *Automated Tools for Information System Design*, H. J. Schneider and A. I. Wasserman (eds), North-Holland Pub., 1982.

[Batini and Lenzerini 83] C. Batini and M. Lenzerini, " A Conceptual Foundation to View Integration," *Proc. of the IFIP TC.2 Working Conference on System Description Methodologies*, Elsevier, Amsterdam, 1983.

[Batini et al 86] C. Batini, M. Lenzerini, and S.B. Navathe, "A Comparative Analysis of Methodologies for Database Schema Integration," *Computing Surveys*, Vol. 18, No. 4, December 1986.

[Ceri 83] S. Ceri (ed), "Methodology and Tools for Data Base Design," North-Holland Pub., Amsterdam, 1983.

[Chen 76] P. Chen, "The Entity-Relationship Model: Towards a Unified View of Data," *ACM Transactions on Database Systems*, Vol. 1, No. 1, 1976.

[Cornelio and Sheth 86] A. Cornelio and A. Sheth, "The Schema Integration Tools - User's Manual," Internal Memo, Honeywell CSDD, August 1986.

[Dayal and Hwang 84] U. Dayal and H. Hwang, "View Definition and Generalization for Database Integration in Multibase: A System for Heterogeneous Distributed Databases," *IEEE Trans. on Software Engineering*, Vol. SE-10, No. 6, November 1984.

[de Souza 86] J. de Souza, "SIS - A Schema Integration System," *Proc. BNCOD5 Conference*, 1986.

[Elamsri and Navathe 84] R. Elmasri and S. Navathe, "Object Integration in Logical Database Design," *Proc. of the 1sr Intl Conf. on Data Engineering*, Los Angeles, April 1984.

[Elmasri et al 85] R. Elmasri, A. Hevner, and J. Weeldreyer, "The Category Concept: An Extension to Entity-Relationship Model," *Data and Knowledge Engineering*, Vol. 1, June 1985.

[Elmasri et al 86] R. Elmasri, J. Larson, and S. Navathe, "Schema Integration Algorithms for Federated Databases and Logical Database Design," *submitted for publication*, 1986.

[Larson 86] J. Larson, "A Visual Approach to Browsing in a Database Environment," *Computer*, June 1986.

[Larson et al 87] J. Larson, S. Navathe, and R. Elmasari, "Attribute Equivalence for Schema Integration," to appear in *IEEE Transactions on Software Engineering*, 1987.

[Lundberg 82] A. Lundberg, "Information Modeling Tool," in *Automated Tools for Information System Design*, H. J. Schneider and A. I. Wasserman (eds), North-Holland Pub., 1982.

[Mannino and Effelsberg 84] M. Mannino and W. Effelsberg, "A Methodology for Global Schema Design," Tech. Rep. No. TR-84-1, Comp. & Ino. Sc. Dept, Univ. of Florida, September 84.

[NAvathe and Awong 87] S. Navathe and A. Awong, "Abstracting Relational and Hierarchical Data with a Semantic Data Model," *Proc. of the 6th Intl. Conf. on Entity Relationship Approach*, November 1987.

[Navathe and Gadgil 82] S. Navathe and S. Gadgil, "A Methodology for View Integration in Logical Database Design," *Proc. of the Eight International Conference on Very Large Databases*, Mexico City, September 1982.

[Navathe et al 86] S. Navathe, R. Elmasri, and J. Larson,"Integrating User Views in Database Design," *IEEE Computer*, Vol. 19, No. 1, January 1986.

[Navathe et al 84] S. Navathe, T. Sashidhar, and R. Elmasri, "Relationship Merging in Schema Integration," *Proc. of the Tenth International Conference on Very Large Databases*, Singapore, August 1984.

[Reiner et al 84] D. Reiner, et. al., "The Database Design and Evaluation Workbench (DDEW) Project at CCA," in *IEEE Database Engineering*, Vol. 7, No. 4, pp 34-39, December1984.

[Teorey and Fry 82] T. Teorey and J. Fry, Design of Database Structures, Prentice-Hall, Englewood Vliffs, NJ, 1982.

[Tsichritzis and Lochovsky 82] D. C. Tsichritzis and F.H. Lochovsky, *Data Models*, Prentice-Hall, 1982.

[Wiederhold and Elmasri 79] G. Wiederhold and R. Elmasri 79, "A Structured Model for Database Systems," Rep. STAN-CS-79-722, Computer Sc. Dept., Stanford Univ., Stanford, CA.

[Yao et al 82] S. Yao, V. Waddle and B. Housel, "View Modeling and Integration Using Functional Data Model," *IEEE Trans. on Software Engineering*, Vol. SE-8, No. 6, 1982.