Architecture for Dynamic Schema Evolution in Heterogeneous Database Environments: A Prototype System and Its Evaluation

Working paper #2003-10

G. Shankaranarayanan

Boston University School of Management
Information Systems Department
595 Commonwealth Avenue
Boston, MA 02215

gshankar@bu.edu
Phone: (617) 353-4605
Fax: (617) 353-5003
Abstract

Dynamic schema evolution is the process of evolving a database schema by incorporating changes in a timely manner, without loss of existing data, and without significantly affecting the day-to-day operations of the database. Systems that manage schema evolution are described in the literature, but address schema evolution single, stand-alone, object-oriented databases. Organizations typically use an integrated set of multiple different databases for satisfying their complex data needs. Managing schema evolution in such heterogeneous data environments (HDE) has not been dealt with. A logical architecture for managing dynamic schema evolution in a HDE is proposed in this paper. The architecture incorporates a graph-theoretic framework that is based on a set of requirements identified for dynamic schema evolution in a HDE. Its implementation in a prototype software system (SEMAD) is described. Implications for automating dynamic schema evolution are examined using SEMAD. An exploratory case study for evaluating the usefulness of SEMAD (to database administrators) in dynamically managing schema changes in a HDE is also presented.

1. Introduction

A database schema is the description of a database [9]. It changes and evolves during design and continues to do so throughout the operational life of the database [25]. A study by Sjoberg reveals that schema changes are as common in the operational phase of a database life cycle as during the design phase [22]. The study examined changes to the schema of a relational database used in health management. In the 5-month period of design and development 41 new relations were added and 24 removed (65 changes in all). There were 470 attribute changes (additions and deletions) in this period. During 13 months of observation when the database was operational, 107 relations were added and 92 deleted (299 changes in all). The number of attribute changes was 2324 during this period.

Schema changes are necessary for the following reasons: (1) the real world represented by the database changes over time and the schema has to change to be consistent with it. (2) Users’ perceptions and requirements change and the schema must change to accommodate them. (3) The application domain modeled by the database can change and the database schema has to be adapted accordingly. Schema evolution is the process of changing the database schema while maintaining its consistency without significantly affecting the daily operations of the database. It implies evolving the schema from one consistent state to the next by incorporating the changes and ensuring that the existing data is modified to be consistent with the evolved schema.

Reorganizing the database is the solution for managing schema changes and involves moving part or the entire database “off-line”. Reorganizing cannot be performed frequently as it deprives the users access to part or the entire database during reorganization. If performed infrequently, the database schema is inconsistent with its world of reference during the intervals between reorganizations. It is best that schema evolution be dynamically managed. Dynamic schema
evolution is defined as the ability of the database to incorporate changes to its schema in a timely manner, with minimal modifications to existing dependent applications, and without significantly affecting the day-to-day operations of the database. The impact of schema evolution has been examined in several application domains such as office information systems [10][12], personal databases [6], engineering design databases [1][4], and real-time manufacturing databases [26]. A large body of literature exists reflecting the extensive work completed in this area.

Organizations typically depend on an integrated, inter-related, heterogeneous collection of databases for their decision support requirements. A change to the schema of any one of these databases affects that schema and other database schemas in this inter-related set, affects the data extracted from this set, and significantly impacts decision-making. Managing dynamic schema evolution is complex and difficult in such heterogeneous database systems. In this paper the term heterogeneous database environment (HDE) is used to describe a data system that consists of different databases, storing different types of data, and managed by different types of database management systems (relational, object-oriented, object-relational, and even flat files accessed by programs in 3GL). A global or a federated schema is often used to describe the structure of a HDE and helps link together all of the underlying databases. Consider a large administrative HDE in a university consisting of financial, research projects, employee, and student databases. A new real-estate database (information on building(s) owned / leased / rented by the university, including office and classroom space) needs to be integrated. This triggers changes to the global schema because income/expenditure associated with each building should be tied to financial data, research labs in buildings linked to projects, and faculty/departments associated with buildings where they are housed. A schema change can impact not only other parts of that schema but also the schema of other databases in the HDE as seen above. Such cascading
changes must be completely and correctly tracked to maintain the HDE consistent. Manually performing these tasks is difficult because of the complex inter-relationships that typically exist in a HDE. Automated management of dynamic schema evolution is hence necessary. Research has addressed dynamic schema evolution but in single, standalone databases. Schema evolution in a HDE has not been studied. Specifically, dynamically managing schema evolution, implications for automating it, the extent of automation possible, and the design and implementation issues for a system to automate dynamic schema evolution in a HDE have not been examined. Our first objective in this paper is to propose an architecture for automated management of dynamic schema evolution in a HDE. The second is to describe the implementation of this architecture in a prototype software system called SEMAD (Schema Evolution Management Advisor), a decision support system for dynamic schema evolution and to examine the implications for automating it. The final objective is to describe an exploratory case study to evaluate the effectiveness of SEMAD in assisting database administrators.

The remainder of this paper is organized as follows. Section 2 summarizes relevant literature focusing on systems for schema evolution and defines the scope of this paper. The logical architecture for dynamic schema evolution is proposed in section 3. Its implementation in SEMAD is described in section 4 and implications for automating dynamic schema evolution are discussed here. Section 5 describes the exploratory case study used for evaluating the usefulness of SEMAD. Concluding remarks are presented in section 6.

2. Relevant Literature

Literature in schema evolution may be viewed from two perspectives: the activity managed and the data model used. Based on activity, there are three well-defined and inter-related topics: core schema evolution, version management, and application management. Core
schema evolution has two parts. *Schema evolution* involves identifying and incorporating schema changes without affecting its consistent state and *database evolution* (or *propagation of schema changes*) involves changing the data in the database to conform to the evolved schema [25]. The preferred approach to schema evolution is defining a consistent schema using a set of schema invariants and rules to preserve consistency. SEMAD implements this using a semantic model and a graph-based framework. The preferred approach to database evolution is coercing the affected data to conform to the changed schema using conversion, screening, filtering, or a combination of these methods. The conversion method modifies the existing data to conform to the evolved schema and results in only one version of data that is consistent with the current schema. If a schema change causes the removal of a schema object (e.g. an entity class) then the data corresponding to this object (e.g. a table corresponding to the entity class) is “lost”, i.e., it is no longer accessible from this evolved schema. Screening and filtering methods map (using functions) data between the old and evolved schema. Data corresponding to a deleted schema object is still visible when accessed through the old schema and not visible (or has default values defined by the map) when accessed through the evolved schema. Screening and filtering are useful in applications such as engineering and microchip design where users need multiple versions of both the schema and data. In typical business applications multiple versions of schema and data are not maintained and conversion is sufficient for database evolution. SEMAD adopts the conversion method to implement database evolution. The second topic based on activity managed, version management, deals with maintaining and managing multiple versions of the database schema and data associated with each. Application management, the third topic based on activity managed deals with how an application may be maintained such that it can access the data despite changes to the schema upon which it is dependent. Each is a very
extensive area of research on its own and is outside the scope of this paper. Based on the second perspective, the data model used, three models may be identified: the object-oriented, relational, and conceptual models such as Entity Relationship model (ERM). SEMAD manages schema evolution using a conceptual model that is an extension of the ERM [7].

Among systems for schema evolution described in literature, the most relevant ones are GemStone [16], Encore [23], ORION [3], OTGen [13], O₂ [27], Tigukat [17], TSE [18], and Frandole-2 [2]. GemStone focuses on class evolution in object-oriented databases by defining invariants to manage schema changes [16]. The database evolution is by conversion. Encore emphasizes object type evolution and focuses more on the database evolution using filtering than on schema evolution [23]. The set of schema changes examined in both the above is limited in that each focus on a specific subset of the schema (class in GemStone and type in Encore). ORION implements comprehensive schema and database evolution in object databases [3]. It includes a complete taxonomy of schema changes and a set of invariants to manage the semantics of schema changes. Database evolution in ORION is implemented using a method similar to screening. OTGen is similar to ORION in that it implements schema evolution using invariants and database evolution using screening [13]. O₂ defines structural and behavioral consistencies to manage evolution of objects and object types [27]. O₂ points out the need for involving the designer in schema evolution and raises the issue of automating schema changes. Tigukat uses a set of axioms for implementing schema evolution and does not address database evolution explicitly [17]. It is the first to address dynamic schema evolution and the axiomatic model is developed with this intent. View mechanisms to assist object schema evolution by creating personal schemas from a base schema are deployed in the TSE system [18]. The governance for schema evolution is by rules built into the view generation algorithms. Changes
are propagated to object instances using a "filtering-like" method. Frandole-2 is a system for schema evolution in object databases and employs an ER-like model to describe the entire schema or context [2]. The semantic context is an abstraction that permits regrouping of certain elements in the schema and is represented in Frandole-2 as a graph that is similar to an ERM.

These systems are based on the object model and work with a single, stand-alone object-oriented database. SEMAD manages dynamic schema evolution in a HDE that may include object-oriented and other types of databases. Dynamic schema evolution in SEMAD is implemented using a framework that is based on a semantic model. The schema of the HDE is first represented as a semantic model that is then mapped onto a graph. A set of graph-based rules defines its consistent state in SEMAD. A comprehensive taxonomy of schema changes is identified based on the semantic model representation and its corresponding graph. The taxonomy helps define the steps for incorporating and understanding the implications of each schema change. A set of graph-based rules defines the consistent state of the schema. Graph-based operators and algorithms implement the changes and manage schema consistency when changes are made. SEMAD uses conversion for database evolution. It is a decision support tool to assist the DBA in the management of dynamic schema evolution in a HDE.

3. Architecture for Dynamic Schema Evolution

To dynamically manage schema evolution in a HDE the following requirements need to be addressed. The HDE consists of different types of databases and it is beneficial to adopt a representation scheme that is not dependent on the database type. All possible changes to the schema must be identified. The implications of each change and all of the resulting cascading changes must be known and understood with respect to each database schema and the HDE as a
whole. The consistent state of the schema must be preserved when incorporating the changes. Finally, all of these changes must be propagated to the data in the underlying databases.

The proposed architecture consists of three-layers, each with its own specific functionality. The top layer of this 3-layer architecture is the change management layer and is responsible for storing the schema of the HDE, identifying changes to it, and implementing the changes while preserving its consistent state. A comprehensive framework that is responsible for these functions is embedded in this layer. The middle layer, propagation management, is responsible for database evolution. The bottom layer consists of the databases in the HDE.

**Change Management Layer:** The change management layer incorporates a framework for dynamic schema evolution in the HDE and is responsible for managing schema evolution. The theoretical foundations of the framework are described in an earlier research [20]. It is adapted in this paper for implementation in SEMAD. The components of this framework and how each ties in with the functionality of this layer are described next. The schema of the HDE is represented using a semantic model in this layer. Semantic models are typically used to describe database schemas and offer several advantages. Semantic models are independent of the type of database and hence are best suited for managing schema evolution in a HDE. Semantic models permit explicit modeling of complex real-world relationships that are not supported by relational models and are hidden in object methods in OO models. More information about the world of reference can be explicitly represented and so more types of schema changes can be explicitly identified and managed. Further semantic models can be easily mapped onto a graph. This permits the use of proven graph-theoretic constructs and algorithms to define the framework. In this paper the schema of the HDE is represented using the Unifying Semantic Model (USM) [19]. The first step in the framework is to map this semantic model schema onto its corresponding semantic model
graph (SMG). The USM offers nine distinct class constructs and nine distinct types of relationships. So the SMG has nine distinct types of nodes and nine distinct types of links. The graph SMG is defined as SMG (N, L), where N = \{N_E, N_W, N_I, N_C, N_A, N_D, N_{Ct}, N_{at}, N_M\} is a disjoint union of a set of 9 types of nodes and L = \{L_I, L_W, L_S, L_C, L_A, L_D, L_{Ct}, L_{At}, L_M\} is a disjoint union of set of 9 types of links. To represent inter-attribute (inter-class) constraints, the definition of each constraint is captured as a node of type N_{At} (N_{Ct}) and is linked to the participating attribute (entity class) node(s) using attribute (class) constraint links of type L_{At} (L_{Ct}). The mapping can be used to map a semantic model onto a SMG and map an SMG back to its semantic model.

Changes to the schema can be interpreted as changes to the SMG. To understand all schema changes and identify the implications of each, a comprehensive taxonomy of schema changes is constructed based on the SMG. It rigorously examines all changes possible to each node type and link type in the SMG and identifies the implications of each change. Using this each change can be interpreted as a sequence of steps to be incorporated in the SMG.

A set of thirteen rules is first defined to specify the consistent state of the SMG and its schema. A SMG that satisfies these rules (trivially or otherwise) is said to be in a consistent state. The rules can be used to detect violations to the consistent state of the SMG before, during, and/or after changes are incorporated. To incorporate schema changes, a set of basic graph-based operators (to add or remove a node/link from the SMG) is defined. Using the taxonomy the basic operators are selectively grouped to define consistency-preserving (CP) operators, one for each type of schema change identified. Each CP-operator consists of a set of basic operators and when executed atomically will preserve the consistent state of the SMG. The completeness and correctness of the CP-operators has been shown in [20].
Finally, a set of graph-based algorithms supplements the schema evolution framework. These algorithms serve two purposes. First, for a given schema change to be incorporated into the SMG the algorithms help visualize and identify the affected objects (nodes/links) in the SMG. It is an important especially in a HDE as a change in one database schema may cascade and result in a myriad of changes not only in that schema but also in other database schemas in the HDE. Secondly, they help detect inconsistencies (violation of one or more rules) in the SMG even before changes are made. The correctness of these algorithms has been shown in [22].

< INSERT Figure 1: Architecture for Dynamic Schema Evolution HERE>

In this architecture the structure of a HDE represented by a global or federated schema (in a consistent state) is captured in the change management layer (shown as “Existing Semantic Model” in figure 1) and can be mapped onto to a SMG (“Semantic Model Graph” in figure 1) using the mapping scheme. Changes to the database schema along with the action items required to comprehensively manage each change can be identified using the taxonomy. Parts of the HDE that may be affected by a change and all its cascading changes can be identified using graph-algorithms before incorporating the change(s). The changes can be incorporated into the SMG using the set of CP-operators. Inconsistencies that arise in the SMG (and schema) when making changes can be detected using the graph algorithms and resolved (with DBA assistance). A changed (or evolved) SMG can be translated back into its corresponding evolved schema (“Evolved Semantic Model” in figure 1) that is guaranteed to be in a correct and consistent state. The final step is to propagate the schema changes to the set of underlying databases in the HDE.

A subset of the schema for a HDE used to manage the Global Climate Change and Hydrology Data (GCCHD) is used as a running example in this paper to illustrate key concepts. Scientists and hydrologists use this HDE to investigate and understand the effects of various
phenomena on the earth's climate such as global warming and the green house effect. This HDE contains facts about atmospheric properties such as cloud characteristics, profiles of temperature, pressure, and humidity, vegetation, radiation, and run-offs. The data is measured by using various instruments (probes) that are mounted on satellites, aircraft, and balloons. Observation stations distributed across the globe also measure and record some of this data. The HDE also includes data on precipitation and water flow (peak and mean-daily-discharge) measured by precipitation stations or gauging stations. The USM representation of the global schema for this HDE is shown in figure 2. A weak class is represented using a box with double borders, complex class using a box with dark borders, and simple strong class using a box with single-line border. The subclass relationship is represented using a hexagon with the letter 's'. Attributes are represented by ellipses. Domain classes are represented by rounded rectangles. For example, "Zenith" captures the angle of a satellite probe with reference to the sun. It is an attribute of the class "Satellite-Probes" and draws its value from a domain of real numbers. Only a small subset of attributes and domains is shown. The SMG obtained by mapping the semantic model in figure 2 onto a graph is shown in figure 3.

<INSERT Figure 2: Semantic Model Schema for GCCHD HERE>

<INSERT Figure 3: Semantic Model Graph (SMG) for GCCHD HERE>

Propagation Management Layer: The middle layer of the architecture, propagation management, manages the metadata (called Propagation Map or PM) required to propagate schema changes made to the SMG in change management layer to the heterogeneous set of databases (in database layer). In a global/federated schema model representing more than one database, a change in the schema must be identified not only in terms of the database object(s) it represents, but also in terms of the specific database(s) to which each database object belongs. So
each database object is mapped to the semantic model object that represents it and to the
database in the HDE that the database object belongs to. PM stores and manages this mapping
data. The PM also includes data on the types of databases in the database layer and about the
DBMS that manages each. When changes take place to the schema in the change management
layer and/or to the data objects in the database layer, the PM must be updated to reflect these.
The arrows between the layers in figure 1 represent these interactions. The translator is a
component in this layer that reads the mapping information and informs the administrator. It
helps communicate the schema changes to the administrator in terms of changes required to the
database objects. It also generates data definition statements to implement schema changes in the
set of databases. Changes to the SMG and to the database(s) must be reflected in the mapping
information.

Using this architecture a schema change can be specified in two ways: directly in the
semantic model (adding, deleting, or modifying a semantic model object) or to the database
objects (adding, deleting, or modifying). For the first case, in the change management layer, the
implications of the change can be identified, and the change(s) can be incorporated into an
evolved schema while guaranteeing its consistent state. Using the PM in the propagation
management layer, the database objects that require changes and the corresponding databases
where the changes should be made can be tracked. The changes can then be propagated to the
database objects in the database layer. When a change occurs at the database layer, it can be
interpreted in terms of the semantic model using the PM in the propagation management layer.
From this point on, managing this change follows the same sequence as in the previous case. The
architecture allows tracking changes to the semantic model and to the databases.

4. Implementing Dynamic Schema Evolution
SEMAD implements the logical architecture and is used to examine the implications of automating dynamic schema evolution. The GUI for SEMAD is a modeling tool for designing and managing semantic models. It consists of a tool bar and a drawing canvas. Users may define a new construct (entity class or relationship class) by selecting the appropriate button from the tool bar and dragging it on to the canvas. The modeling tool is capable of storing several semantic models as projects. It supports a menu driven system for project management functions such as defining a new, opening an existing, saving a completed, and closing an open project. The system also supports a number of other functions including the ability to clear the drawings, delete specific model objects, and scrolling. Figure 4 shows the GUI in SEMAD displaying the GCCHD model.

4.1 Implementing the Change Management Layer

In SEMAD, the semantic model representing the HDE-schema is captured in the change management layer using a repository for model metadata. The repository accurately reflects the structure of the semantic model using a relational database with a set of 22 relations designed for this purpose. It is managed by a Sybase System 11 RDBMS and is linked to the GUI using Sybase database library functions embedded in C language. As the model is created or modified the metadata such as the name of the model-object, its purpose, participating classes and cardinality (for relationship objects) are obtained from the user using interfaces. A software module reads the model information from the repository and creates a set of adjacency matrices to represent the graph SMG. The SMG contains (9) different types of links and each link type must be distinguishable in the adjacency matrix. So the matrix is modified to include entries besides 0 and 1. As the graph SMG is a directed graph, the matrix is not symmetrical. Using one adjacency matrix to represent the SMG results in a very large and sparsely populated matrix. A
solution is to use a set of smaller representations instead. Besides preserving all of the
information in the original matrix this results in a more manageable set of data structures
consisting of:

- **PAM** or the **primary adjacency matrix** to capture the interaction, weak, sub-class, and
  complex relationship type along with the entity classes associated.
- A linked list of attributes for each entity class and a domain matrix to capture the domain
  associations of the attributes
- One matrix each to capture the attribute constraints and the class constraints.
- One matrix to represent the composite/member relationships.

The consistency preserving operators (CP-operators) for schema changes and the set of
algorithms reside in the *change management* layer. When a schema change is identified, an
algorithm (Find-Domino-Vertices) is used to determine if other parts of the SMG are affected by
this change and if so, it returns the set of affected graph-objects as the candidate set. To detect if
these changes (initial and cascading changes if any) result in disconnecting the SMG (causing an
inconsistent state), the Find-Cut-Vertex and Find-Cut-Edge algorithms are used. These
algorithms read the graph from its matrix representations. If no inconsistencies are detected, then
the appropriate graph-based schema change operators are used to incorporate the change into the
SMG. Each operator (function) reads the graph SMG from the set of data structures and makes
the change by modifying the appropriate member(s) in this set. The operators also update the PM
in the *propagation management* layer to reflect the changes made. Should the algorithms or
operators detect inconsistencies to the SMG, SEMAD informs the administrator of the error and
requests assistance. Pop-up interfaces are used to display the error/warning messages (samples
shown in figure 5). The system modules that convey these messages interact with the metadata
repository and with the PM (in *propagation management* layer) to interpret the errors/warnings
in terms of database objects in the *database* layer. When the initial schema change and all
cascading changes are completely incorporated, these are then propagated to the corresponding
database objects and associated databases using the PM.

<INSERT Figure 5: Sample interfaces in SEMAD HERE>

Consider for example, a smaller subset (considering the atmospheric data only) of the
GCCHD schema and semantic model shown in figure 2. The subset is centered on the complex
entity class \textit{Data} and includes its base classes, \textit{Clouds} and \textit{Surface-Conditions}. The other classes
included are \textit{Cloud-Characteristics}, \textit{Atmospheric-Properties}, \textit{Regions}, \textit{Countries}, \textit{Continents},
\textit{Probes}, \textit{Probe-types}, \textit{Satellite-Probes}, \textit{Aircraft-Probes} and \textit{Field-Programs}. The primary
adjacency matrix corresponding to the SMG for this subset is shown in table 1.

<INSERT Table 1: Adjacency Matrix at the CM-Layer for GCCHD HERE>

\subsection*{4.2 Implementing \textit{Propagation Management} Layer}

This layer assists in propagating schema changes to the databases. The propagation map
(PM) in this layer includes two types of mapping information are: one to map the semantic
model (schema) object to its corresponding database object and the other to associate each
database object with its corresponding database(s) in the HDE. These are managed using two
sets of relations: Schema-Database object map (SD) and the Database object-Database map (DD)

\textbf{Mapping Schema Object to Database Object}: Schema-Database map contains information
that maps the semantic model schema objects to the database object(s). Schema (semantic
model) objects are represented as tables/files/objects at the database level. Relationships in the
schema may be represented in the database(s) as an individual database object with its own
identity or may be part of the information in database objects corresponding to schema objects
that participate in this relationship. To implement dynamic schema evolution we need this
information to propagate changes made to schema object(s). The SD-map information is an
ordered pair $<s, d>$ where $s$ is the name of the object in the schema and $d$ the name of the corresponding database object. In the heterogeneous database system, a single database object may be split/replicated and stored in two or more databases. If the schema-object (entity) corresponding to this database object is changed all distributed and/or partitioned copies of the database object need to be modified to maintain consistency.

**Mapping Database Object to Database:** The database object to database map (DD-map) maintains the mapping information on how the database objects (tables/files/objects) corresponding to a given schema are distributed and stored in the HDE. A database object (e.g. a relation) that corresponds to an entity class in the schema may be partitioned vertically (or horizontally) and stored in more than one database. To implement database evolution, the set of one or more databases to which changes should be propagated must be identified. The DD-map is an ordered pair $<d, D>$ where $d$ is the database object and $D$ the database in the HDE where it is stored. If a database object is replicated in the HDE then multiple ordered pairs are used to capture the replication. If the database object is vertically partitioned, then $d$ includes the subset of attributes as well. In SEMAD the two maps (SD and DD) together associate a schema change made in the *change management* layer with the database objects and the databases in which the objects reside in the *database* layer. The translator generates data/object definitions to implement the change in the appropriate database(s). The editable statements are displayed using interfaces and implementation proceeds only after the administrator approves it. Upon completion, the SD and DD maps are updated to reflect the changes. The PM is thus consistent with the schema and the database(s) in the HDE that the schema represents at all times.

*<INSERT Table 2: Propagation Map (PM) for GCCHD example HERE>*
PM captured in the *propagation management* layer for the GCCHD example is shown in table 2. Even though SEMAD has the ability to dynamically manage the process in a truly heterogeneous database environment, this example uses a HDE that consists of relational databases. The GCCHD schema shown in figure 2 is captured as a set of relations distributed over 4 relational databases. Table 2 shows the PM including the SD and the DD maps corresponding to the subset of the GCCHD schema described at the end of section 4.1.

Consider the schema change, deleting the entity class *Probes* from the GCCHD system, to illustrate how the change and propagation management layers help implement dynamic schema evolution. The change may be initiated by dropping the database object corresponding to *Probes* or by removing the schema object *Probes* from the GCCHD schema. In the former case, using the mapping information, the change can be equated to the latter case. Though the architecture tracks changes initiated from either direction, at this time, SEMAD is capable of tracking changes initiated from the schema. In the graph (SMG) corresponding to GCCHD stored in the change management layer, the change requires the removal of the node corresponding to *Probes*. The algorithm Find-Domino-Vertices identifies the set of nodes

\{Probes, Satl_P, Aircr_P, Fld_pgm, Probe-Types, Regions, Countries, Continents, Data\} \(^1\) in the SMG affected by this change using the concept of “reachability” in graphs starting with the initiating node *Probes* (shown in figure 6).

<INSERT Figure 6: Interface in SEMAD showing objects affected HERE>

Removing *Probes* implies removing its dependent complex class *Probe-Types* and the ternary relationship *Measure* that is dependent on *Probe-Types*. The algorithm Find-Cut-Edge reveals

---

\(^1\) *Satl_P, Aircr_P,* and *Fld_pgm* are dependent subclasses of *Probes*. *Probe_Types* is a dependent complex class of *Probes*. Removing *Probe_Types* implies removing ternary interaction relationship *Measure*. This affects *Regions* and *Data*. *Countries* is a dependent complex class of *Regions* and *Continents* a dependent complex class of *Countries*. 
that removing this link corresponding to *Measure* in the graph SMG will result in disconnected components (one with the node corresponding to *Regions* and its dependent complex class and the main part with *Data* and its base classes) in the graph causing an inconsistent state. The database administrator (DBA) is informed of this and is requested to intervene. The DBA has three options to preserve consistency – to reject the change completely, to remove the component with *Regions* completely, or to define a new binary interaction relationship link between *Data* and *Regions* thereby permitting the removal of *Measure*. As the algorithms are “looking-ahead” to identify problems, no action is required in the first case. In the second case, SEMAD consults the mapping information in the propagation management layer to locate the database objects corresponding to the nodes and links in the graph that are to be removed and the databases in the HDE in which these reside. These objects are deleted after DBA confirmation. The PM (DD-map) in the propagation management layer is updated to reflect the removals. The semantic model objects are then removed and the PM (SD-map) is updated to reflect these changes.

If the third option was indicated by the DBA then a new link is added to the GCCHD graph. The database object(s) corresponding to the new relationship is defined in the databases – the names and locations are specified by the DBA. The PM (SD and DD maps) is updated to reflect these additions. Having evolved the schema and databases while preserving consistency, the nodes corresponding to *Probes, Satl-P, Aircr-P, Fld-pgm* and *Probe-Types* together with the associated subclass, complex, and interaction relationship links including *Measure* (that can now be deleted without violating consistency) are removed in the same way as described in the previous paragraph under the second option. The resulting adjacency matrix following this change (and implemented using the third option) is shown in table 3.

<INSERT Table 3: Adjacency Matrix for GCCHD after removal of *Probes* HERE>

4.3 Implications for Automation
The above example reveals some implications for automating dynamic schema evolution in a HDE. First, examining the semantic model makes it easier to identify cascading changes more comprehensively. Visualization improves decision-making by presenting complex information in a more intuitive fashion and that is easier to digest. A visual interface (not necessarily a conceptual model) should be part of any automated tool for dynamic schema evolution in a HDE. Second, a single schema change could result in a myriad of cascading changes. Identifying and understanding these changes comprehensively is best performed with automated support especially in a HDE that is typically managed by more than one DBA. Third, when a database (or a part of it) becomes disconnected from the rest of the HDE resulting in an inconsistent state then consistency must be restored. As seen in the example above, this involves adding one or more new schema object(s) or deleting the entire disconnected part. In the former case, the new object may represent an entity or a relationship that does not exist or is incorrect in the world-of-reference of this database. This compromises the accuracy of the database. Whether the object is truly reflects relevant/accurate information in its world of reference or not can only be determined by an expert user / DBA. Further, only the DBA can decide between the alternate implementation choices (which database(s) in the HDE, whether replicated, what must it be named, what is the identifier/key attribute set, etc.) for creating the newly added object(s). If only one version of the schema and data is maintained then deleting objects in the schema to preserve consistency implies losing the data (in the HDE) corresponding to these objects. The burden of deciding whether the data should be retained (by rejecting the change or adding new objects to preserve consistency) or not rests with the users and/or DBA. Hence dynamic schema evolution can at best be semi-automated. Four, DBAs typically manage changes (at least the simple ones) by directly changing the data and database object(s). They may perceive automated tools (such
5. Evaluating SEMAD

To manage schema evolution in a HDE the database administrator(s) must be able to identify schema changes and identify all the cascading changes triggered by each. This is a challenge given the complex set of inter-relationships amongst the multiple databases in the HDE. The administrators must also completely incorporate all the schema changes. Oftentimes some cascading changes are overlooked and later pointed out by users. The evaluation process specifically attempts to answer the following questions.

- Is the task of comprehensively identifying schema changes easier with SEMAD?
- Is the task of incorporating schema changes easier with SEMAD?
- Overall, does SEMAD ease database administration in managing dynamic schema evolution?

5.1 Empirical Methodology

SEMAD was evaluated for its usefulness using an exploratory case study. Adopting multiple methodologies is often recommended for validating human-computer interaction [18]. The multi-methodological scheme for validation includes prototyping, case study, and experimentation. Benbasat et al. point out that a case study approach is suitable for capturing the knowledge of practitioners [6]. As one of the objectives in this evaluation was to understand how the DBA manages schema changes in a HDE and to incorporate this knowledge to improve SEMAD, the exploratory case study appears appropriate. The study was conducted using the
GCCHD system (with only relational databases) described in this paper and structured (units of analysis, data collection, etc.) following Yin's recommendations (stated in [26]).

**Units of Analysis:** Units must be familiar with the HDE used and must be experienced in maintaining databases. Familiarity was important as a lack of it may lead to bias. Also, since SEMAD uses semantic models, the unit of analysis must have a good understanding of some (at least one) semantic model. The unit of analysis is therefore one (or two individuals who might work together) meeting these requirements: (a) a sound knowledge of the HDE used in the study, (b) experienced in maintaining the database(s), and (c) having a good understanding of semantic models. Two administrators (one from Atmospheric Science and one from Hydrology) were chosen. They would work together as one unit (referred to as DBA in the rest of this section).

**Tasks:** Two sets of identical tasks were defined based on the GCCHD system. Each task simulated a schema change from the set of changes observed in this HDE over a one-year period. The set of tasks were chosen such that all possible schema changes (identified in the taxonomy based on the semantic model – section 3.1) are represented. The DBA was asked to perform each task in the first set manually and then perform each task in the second set with SEMAD using its GUI displaying the semantic model schema of the HDE. This sequence ensured that there were no learning effects that could bias subsequent responses from the DBA. The DBA was also not shown the semantic model until after the completion of the first set of tasks. The order in which the tasks were performed was identical in both sets. The tasks are listed and described in table 4.

*<INSERT Table 4: Description of Tasks HERE>*

**Format for Data Collection:** Each task in the first set required them to manually perform one or more changes to the HDE. For each change, the DBA was asked to manually examine and identify possible changes and cascading changes (if any) and suggest solutions to incorporate
them. The DBA was also requested to explain the rationale behind each solution. Upon completion of each task, the count of the number of changes suggested by the DBA was recorded along with the methods and solutions suggested to incorporate the changes. The DBA was then asked to perform the second set of tasks, this time using SEMAD. Upon completion of each task, the count of the number of changes identified and the solutions suggested were again recorded. SEMAD makes each change once in the schema and once in the database(s) while the DBA suggested changes to the database only. To account for this, only one set of changes was counted for the SEMAD-assisted tasks. The DBA was also asked to rate the perceived usefulness of SEMAD for managing schema changes. This information was also recorded.

**Analysis of Data Collected:** The results from the two sets were compared for the number of changes suggested and the quality of the suggested solutions. Quality of solutions was evaluated using two measures: (1) **Measure of Completeness** that evaluates whether the solutions were comprehensive and included all relevant changes resulting from the specific change addressed. (2) **Measure of Correctness** that evaluates whether the solutions to incorporate changes were correct and preserved the consistency of the HDE. An independent expert was asked to compare the solutions and rate the quality. A senior data quality administrator was chosen to serve as the expert and critically examine the solutions. The measure of completeness was rated on a scale of 1 (most comprehensive) to 5 (least comprehensive). Correctness was rated on a scale of 0 to 2: 0 if no inconsistencies resulted, 1 if the resulting inconsistency is trivial (incorrect but has little or no impact on the HDE), and 2 if the inconsistencies are non-trivial. For example, a rating of 1 would be assigned if a table was orphaned when incorporating some change, but the data in the table is not of any value and would be deleted later by some cleanup process). The two measures are independent of each other. The first measures the comprehensive nature of the changes
suggested and their relevance to the schema change specified. The second evaluates if the changes preserved the consistency of the HDE. The sum of the two ratings was used to evaluate the overall quality of the suggested change(s). A smaller total implies a better solution. The DBA was also asked rate SEMAD based on how satisfied the DBA was with using it (for managing schema changes). This, the Measure of Satisfaction, was rated on a scale of 1 (Very satisfied) to 5 (Very dissatisfied) and was recorded for each task performed.

5.2 Results

For task 1, the DBA suggested adding a relation and a set of attributes. SEMAD helped add an entity to the semantic model schema and then linked the entity with the rest of the schema by prompting the DBA. The corresponding database objects were then added to the HDE. The solution using SEMAD was rated 1 because it helped link the new entity to the rest of the schema and preserved consistency. The manual solution was rated 4 (2 + 2 for the inconsistency observed). SEMAD suggested 3 changes (new class, new attributes, and new relationship) and the DBA suggested 2 (attributes and class only).

In task 2, adding subclass definitions, each solution involved 2 changes. The expert rated both solutions 1 but felt that the process suggested by SEMAD was longer. This was also reflected in the perceived satisfaction of the DBA (rating of 3). For task 3 the DBA suggested adding an attribute to the existing relation for Stations to define subclasses. The solution using SEMAD involved creating two new relationships and corresponding data and then deleting the existing old relationship. Both solutions were rated 1 for completeness and 0 for correctness. The number of changes was 1 and 3 for the manual and SEMAD-assisted tasks respectively. As using SEMAD required more work, the DBA assigned a satisfaction rating of 2.
For task 4 the DBA suggested 2 changes and SEMAD 3. The additional change suggested by SEMAD required the administrator to define attributes to capture the complex relationship correctly. As this solution linked the new relation with the existing HDE it was rated 1 and the manual solution was rated 2. For task 5 both solutions were evaluated as being similar and each involved 2 changes. Task 6 required adding a new complex class by redefining an existing hierarchy. Here SEMAD was found to be deficient. The number of changes suggested by SEMAD was 6 as compared to 3 (adding new data on watersheds by defining table and attributes and relating this information to the existing information on basins). The process suggested by SEMAD was considered “more involved” by the DBA as it required deleting and recreating the classes (satisfaction rating of 4). The expert felt it was a “brute-force” method and assigned a rating of 2 overall. The administrator's solution was rated 1. In the final task of deleting an object from the HDE, SEMAD was able to identify all related information that may be affected and implement the deletion process in a systematic manner. Hence its solution was rated 1 for overall quality. The results are summarized in table 5.

<INSERT Table 5: Summary of Results HERE>

6. Discussion and Conclusions

From analyzing the results of the study and from the experience of conducting the study several observations can be made. When simple changes required, such as the addition of a new entity class, the types and quality of changes proposed by SEMAD are comparable to those identified by experienced administrators. The changes were identified more completely in that cascading changes (such as linking a newly created table with the rest of the database) that are often overlooked were highlighted for the administrators. However, administrators often propose superior and more elegant solutions to incorporate the changes. The solutions from
SEMAD are "bookish" and lack elegance. This is explicitly evident in the tasks that required modifications (redefining existing database objects) and was reflected in the satisfaction ratings (3, 2, and 4). This is attributable to the fact that the framework in SEMAD for managing schema changes implements a modification as a sequence of addition and deletion resulting in more changes and actions needed to manage these. Further, the in-depth knowledge of the domain as well as the experience in database administration can also account for the elegant solutions suggested by the DBA. Eliciting this knowledge from the DBA and creating a knowledge base would improve this aspect in SEMAD.

When the schema change results in a number of cascading changes, SEMAD is able to identify changes more comprehensively as well as suggest more structured solutions. It helps the administrator step through the process of propagating the change(s) to the associated databases. It provides a comprehensive view of all the changes required allowing the administrator plan for better solutions to incorporate these changes as well as understand the impact of these changes on the data. This particularly, is useful in the HDE and is evident from the satisfaction ratings for tasks that involved such changes. The process was very slow as SEMAD maintains information at three layers. For each change, the information in all three layers needs to be updated resulting in the slower processing. To manage schema evolution in a heterogeneous database environment, SEMAD needs to have the semantic model, the metadata associated with the model, and the mapping information that relates the model to the set of underlying databases. Defining this information in SEMAD is a time-consuming task but it is an initial and one-time cost. Subsequent changes are captured incrementally.

There are two areas that benefit from the support offered by SEMAD: (1) identifying all the changes resulting from a specific schema change and understanding how these changes
relate to each other. The first is important for comprehensively managing the changes to maintain the consistency of the HDE and the second for implementing solutions to incorporate the changes. (2) Identifying the database objects that need to be changed so that schema changes may be propagated to the underlying databases in the HDE. This is reflected by the quality ratings (ranging between 1 and 2) for SEMAD-assisted solutions for all the tasks. The administrator can propose superior solutions to implement changes in the database, particularly when modifying existing database objects, as evidenced by the satisfaction ratings. These solutions affect the performance of the database(s) in the HDE. SEMAD does not analyze database performance issues (e.g. normal forms in relational databases) in its recommendations for incorporating changes. Neither does it evaluate if a specific implementation is more suitable for query-driven databases than update-driven databases. The DBA is clearly in a better position to implement the changes in the HDE and must be involved extensively. This supports the observation that the management of dynamic schema evolution in a HDE cannot be fully automated. The DBA must be consulted and must have the option of rejecting/accepting/modifying the solutions that implement changes to the databases in the HDE.

In this paper a three-tier architecture to manage dynamic schema evolution in a HDE has been proposed. It comprehensively supports core schema evolution in a HDE by tracking and incorporating schema changes while maintaining the schema consistent, identifying cascading changes, and propagating changes to the underlying databases. To our knowledge, this research is the first of its kind to address dynamic schema evolution in a HDE. Given that most business information systems include heterogeneous databases, we strongly believe this will assist the database administrator in effectively managing database schema changes. This research critically examines the extent to which the process of schema evolution may be automated.
The research also evaluates the usefulness of a semi-automated system for managing schema evolution using SEMAD using an exploratory case study and the evaluation process is described. Based on the evaluations, it is evident that the system would play a positive role in assisting the DBA. In a HDE, SEMAD appeared particularly useful when changes involved the addition or removal of objects and helped the administrator visualize the possible changes prior to incorporating them and helped comprehensively manage changes.

Dynamic schema evolution has implications for data warehouses. Changes to the schema of the operational/legacy databases would impact the warehouse that is dependent on these databases. A solution would be to define a framework using metadata to manage changes to the schema of source/operational databases and identify the impact of each in on the data warehouse. Metadata in the data warehouse environment includes the logical structure of the warehouse, logical structure of the source databases, and the transformation rules for extracting data from the sources, integrating this data, and populating the data warehouse [10]. We are examining the implications for managing schema changes in the data warehouse environment using an extensive repository of metadata [21].

We are also investigating the applicability of the same model and principles in SEMAD for managing knowledge networks. Knowledge is stored as knowledge objects each being a packet of value-added information that is self-contained and preserves the content and context from its original business setting for reuse in other settings. The knowledge repository consists of several such objects and each of these is related to several other objects in the repository. The inter-relationships represent the fact that the decisions/actions captured in one object form the basis of or were based on decisions/actions captured in other object(s). These objects change over time driven by changes to the content and by changing user requirements. There may exist
several such repositories and organizations to link these in an attempt to integrate and share knowledge and across the organization. The methodology and mechanisms to manage dynamic schema evolution in a HDE are being applied to manage changes in knowledge repositories.

REFERENCES

TABLES AND FIGURES IN THE PAPER (follows the sequence in which these are referred to)

Figure 1: Architecture for Dynamic Schema Evolution

Figure 2: Semantic Model Schema for GCCHD – USM Representation
Figure 3: Semantic Model Graph (SMG) for GCCHD

Figure 4: GUI for Managing Semantic Model Schema (showing GCCHD schema)
Table 1: Sample Adjacency Matrix at the CM-layer for GCCHD example

Superscript and an explanation for the representation

a Data, Regions, & Probe-Types participate in a ternary interaction relationship—captured by cell value 13
b Data is a complex class composed of Cloud and Surface-Condition data—captured by cell value 4
c Probe-Types is a selective collection of Probes—captured by cell value 4
d Satellite-probes, aircraft-probes, and Field-programs are subclasses of Probes—captured by cell value 3
e Cloud-Characteristics is a weak dependent class of Clouds—captured by cell value 2
f Atmospheric-Prop is a complex class dependent on Surface-conditions data—captured by cell value 4
g Countries is a grouping of Regions and Continents a grouping of Regions—captured by entries 4
h Cloud-Characteristics & Atmospheric-Prop participate in a binary interaction relationship—captured by cell value 1
### Table 2: Propagation Maps (SD and DD maps) for GCCHD subset

<table>
<thead>
<tr>
<th>Schema Object -</th>
<th>Database Object Map</th>
<th>Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN Schema</td>
<td>Database Object</td>
<td>In Database(s)</td>
</tr>
<tr>
<td>Clouds</td>
<td>Cloud-data (Type, Top-Temp, Opt-den)</td>
<td>Atmosphere-Lab</td>
</tr>
<tr>
<td>Surface-Conditions</td>
<td>Surface-data (S-Location, S-temp, S-humidity)</td>
<td>Atmosphere-Lab</td>
</tr>
<tr>
<td>Probes</td>
<td>Probes (Probe-Type, Name, Frequency)</td>
<td>Atmosphere-Lab</td>
</tr>
<tr>
<td>Satellite-Probes</td>
<td>S-probes (Name)</td>
<td>Atmosphere-Remote</td>
</tr>
<tr>
<td>Aircraft-Probes</td>
<td>A-probes (Name)</td>
<td>Atmosphere-Remote</td>
</tr>
<tr>
<td>Field-Programs</td>
<td>F-programs (Name)</td>
<td>Atmosphere-Lab</td>
</tr>
<tr>
<td>Regions</td>
<td>Regions (Lat1, Lat2, Long1, Long2, Area, C-name)</td>
<td>Atmosphere-Lab</td>
</tr>
<tr>
<td>Countries</td>
<td>View-on-Countries-Country (C-Name, CT-name)</td>
<td>Atmosphere-Lab</td>
</tr>
<tr>
<td>Continents</td>
<td>View-on-Countries-Continent (CT-Name)</td>
<td>Atmosphere-Lab</td>
</tr>
<tr>
<td>Cloud-Characteristics</td>
<td>Cloud-Char (Type, Motion)</td>
<td>Atmosphere-Lab</td>
</tr>
<tr>
<td>Atmospheric-Properties</td>
<td>Atmo-Prop (Energy-bal)</td>
<td>Atmosphere-Lab</td>
</tr>
<tr>
<td>Probes</td>
<td>P-types - View</td>
<td>Atmosphere-Lab</td>
</tr>
<tr>
<td>Measure</td>
<td>Measure</td>
<td>Atmosphere-Lab</td>
</tr>
<tr>
<td>Determine</td>
<td>CCAP</td>
<td>Atmosphere-Lab</td>
</tr>
</tbody>
</table>

### Figure 6: SEMAD Interface showing affected schema objects

#### Table 3: Primary Adjacency Matrix for GCCHD – after removal of Probes

<table>
<thead>
<tr>
<th></th>
<th>Data</th>
<th>Cloud</th>
<th>Surface-Conditions</th>
<th>Cloud-char</th>
<th>Atmospheric-prop</th>
<th>Regions</th>
<th>Countries</th>
<th>Continents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clouds</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Surface-conditions</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cloud-char</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Atmospheric-prop</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Regions</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Countries</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Continents</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#</td>
<td>Task</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Adding data on Stations to expand atmospheric sciences HDE to include hydrology data (integrating GCC with hydrology data).</td>
<td>Hydrology data includes data on measuring stations (Stations). This task represents adding a new entity class and was performed as an addition of a new relation in the Hydrology database. It also includes defining a relationship between this and the existing data in the GCC environment besides defining the attributes and (if needed) domains. <strong>DBA</strong>: Adds a table to the database with attributes (two changes, define attributes and add a table to HYD database). <strong>SEMAD</strong> adds a new entity class for Stations to the schema. SEMAD gathers metadata on Stations from the DBA and some attributes for it and prompts the DBA for mapping information to define a new relation for Stations. <strong>SEMAD also requests the user to create some relationship between Stations and the rest of the schema to preserve the consistent state of the schema.</strong> After obtaining the mapping information corresponding to the new relationship Monitor, SEMAD creates the new entity class and relationship in the schema and the corresponding relation(s) in the HDE.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Refining information to include the two types of Stations.</td>
<td>This involves creating two new subclasses of Stations, Gauging and Precipitation. As the Stations are described by similar attributes it implies definition of two views. The <strong>DBA</strong> creates an additional type attribute in Stations and creates views corresponding to each of the new station types. <strong>SEMAD</strong> defines two entity subclasses and relationships in the schema and it gets mapping information and defines the views.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Redefining the relationship between Stations and the rest of the HDE</td>
<td>The original relationship between Stations and the rest of the data must be replaced with two new relationships that relate the two subclasses defined. The <strong>DBA</strong> alters the relation Monitor to reflect the relationship between Stations and Stream-Flow data. He creates a relation to capture the relationship between precipitation data and Stations. SEMAD attempts to delete the existing relationship (Monitor) and then create two new relationships. As this disconnects the schema, SEMAD creates two relationships first before deleting the existing one.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Add information on Continents</td>
<td>Continents must be created using existing data on geographical regions by re-defining it and new data on Continents and the countries included in each must be captured. SEMAD adds a Complex class in the semantic model, a complex relationship to link the new Continents with Countries, creates a new relation for Continents and ties it with Countries. The DBA creates a new relation for Continents with latitude and longitude pairs, area, and name as attributes along with a relation to capture the relationship between Countries and Continents.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Create Profile Information</td>
<td>A profile is data on temperature, pressure, or humidity that is measured over different heights above the earth's surface. In our study we include profiles of temperature and humidity only. The task involves creating a set of relations to capture this. Both SEMAD and the DBA adopted the same approach of redefining the view AllData to accomplish this task.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Adding information on Watersheds</td>
<td>A 2-digit Hydrological Unit Code (HUC) identifies a river basin. Each river basin is made of a number of smaller basins (4-digit HUC). Basins are collections of watersheds (6-digit HUC) and these in turn are composed of sub-watersheds (8 digit HUC). Stations are located in sub-watersheds. Prior to this task, the HDE had data that directly related sub-watersheds to river basins. The <strong>DBA</strong> creates a new relation for Watersheds with HUC, name, and Basin-HUC as attributes. The last attribute is used to capture the relationship (Composed of) between Watersheds and Basins. The Sub-Watershed relation was altered to include an attribute for the Watershed it was part of. SEMAD creates a new grouping class for Watersheds, defines a new complex relationship between Sub-watersheds and this, defines another new complex relationship between Watersheds and Basins, and then deletes the existing complex relationship between Basins and Sub-watersheds.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Removing Probe Information</td>
<td>This task was artificially created to test a complicated deletion. It includes deleting all types of objects in the semantic model. The DBA recommends dropping relations corresponding to probe data, Satellite Probes, Aircraft Probes from the Atmosphere-Remote database and field program information from the Atmosphere-Lab database. <strong>SEMAD</strong> attempts deleting the entity corresponding to Probes in the schema and lists the objects affected (figure 11). Upon choosing to continue, SEMAD detects a resulting inconsistency in the schema and warns the DBA (figure 12). After defining a new relationship between Regions and Data to avoid this, SEMAD cleans up the entity classes and relationships from the schema. The associated relations from the database were also removed to complete the process.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Description of case study tasks
<table>
<thead>
<tr>
<th>Task 1: Adding Station information</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of changes</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Measure of completeness (scale of 1 to 5)</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Measure of correctness (scale of 0 to 2)</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Overall Quality rating</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Measure of satisfaction (scale of 1 to 5)</td>
<td>1</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Task 2: Defining subclass information for Stations**

| Number of changes | 2 | 2 |
| Measure of completeness (scale of 1 to 5) | 1 | 1 |
| Measure of correctness (scale of 0 to 2) | 0 | 0 |
| Overall Quality rating | 1 | 1 |
| Measure of satisfaction (scale of 1 to 5) | 3 | N/A |

**Task 3: Redefining relationship involving Stations**

| Number of changes | 3 | 1 |
| Measure of completeness (scale of 1 to 5) | 1 | 1 |
| Measure of correctness (scale of 0 to 2) | 0 | 0 |
| Overall Quality rating | 1 | 1 |
| Measure of satisfaction (scale of 1 to 5) | 2 | N/A |

**Task 4: Adding Continent information**

| Number of changes | 3 | 2 |
| Measure of completeness (scale of 1 to 5) | 1 | 2 |
| Measure of correctness (scale of 0 to 2) | 0 | 0 |
| Overall Quality rating | 1 | 2 |
| Measure of satisfaction (scale of 1 to 5) | 1 | N/A |

**Task 5: Adding Profile information**

| Number of changes | 2 | 2 |
| Measure of completeness (scale of 1 to 5) | 1 | 1 |
| Measure of correctness (scale of 0 to 2) | 0 | 0 |
| Overall Quality rating | 1 | 1 |
| Measure of satisfaction (scale of 1 to 5) | 1 | N/A |

**Task 6: Defining a complex class and modifying complex relationships.**

| Number of changes | 6 | 3 |
| Measure of completeness (scale of 1 to 5) | 3 | 1 |
| Measure of correctness (scale of 0 to 2) | 0 | 0 |
| Overall Quality rating | 3 | 1 |
| Measure of satisfaction (scale of 1 to 5) | 4 | N/A |

**Task 7: Deleting Probe information**

| Number of changes | 6 | 4 |
| Measure of completeness (scale of 1 to 5) | 1 | 2 |
| Measure of correctness (scale of 0 to 2) | 0 | 2 |
| Overall Quality rating | 1 | 4 |
| Measure of satisfaction (scale of 1 to 5) | 1 | N/A |

Table 5: Summary of Results