THE DESIGN OF A SPATIAL DATABASE MANAGEMENT SYSTEM

Qiming Chen
National Land Information System, Beijing, China

ABSTRACT

The design principle of an integrated Spatial DBMS is described. Issues covered by this paper include action capabilities, object-oriented paradigm, virtual object, and the management of check-out environment. This work is issued upon awareness of the weakness of present DBMS in supporting GIS, which may be viewed as a further step to the compromise of database principles and GIS requirements.

INTRODUCTION

The Chinese National Land Information System contains data in raster, vector and tabular forms, captured from present maps, satellite images, photogrammetry films and census, corresponding to 1:1000,000, 1:250,000, 1:50,000, 1:10,000 scale maps. The core of the system is a spatial DBMS.

It is not our intention to discuss in this paper the well established issues such as spatial data structure, vector, raster and tabular data handling, searching algorithms, graphic and image display, ...etc. In fact, most of the spatial DBMS's currently in operation are built with certain loosely coupled components, there lacks an integrated management for a system as a whole, and the transactions utilizing the data are normally out of the system control. Upon awareness of this situation in designing a spatial DBMS we emphasize the following principles :

- The integration of the system.

- Action capabilities of the system for managing application tasks and transforming data processing to data accessing, therefore providing high level user interface.

The corresponding topics we shall discuss in this paper are:
- Action capabilities of the system,
- Object-oriented paradigm,
- Virtual object,
- The management of check-out environment.

ACTION CAPABILITIES

Integrating action capabilities to GIS's is required by providing high level user interface and improving system consistency, since handling application tasks outside the system framework makes it considerably difficult to maintain

data integrity and handle task-oriented constraints. In our approach, facilities for doing so include :

(1) The Trigger Manager [Melk 83a-b], offering a system designer the flexibility of utilizing various modularized actions under various conditions. A trigger is specified in a similar way to an AI production rule, as P --> F , meaning
if (P) then do (F)
Using a specification language, a user can predefine necessary actions to be activated automatically on certain detected system states, or upon execution of certain operations. Thus for example, if a highway under planning is canceled, under the trigger mechanism, the deletion would be propagated to all the coverages where corresponding lines were drawn. The main problem of this approach consists in the computation cost of the alerting processing.

(2) The Process Handler, which may be viewed as a mapping mechanism from a conceptual level task specification (by users) into its implementation (here is a brief introduction, for more detail, refer to author's another paper [Chen 85b]).

A "process" is a linguistic description of a net modelling a task and consisting of a set of decision/action modules with certain conditional linkage; these actions may utilize any domain knowledge or data and carry out update effects to the system state. The specification of a "process" includes :

- a list of actions involved,
- a net definition specifying the internal linkage among the actions and the control flow,
- the linker definitions describing the conditions for each possible path in the net.

Ihis information is given in the form of relations. The net specification system is based on the following concepts :

1. Net-objects, denoting actions, linkers and sub-nets which are represented by expressions, and
2. Net forming operations for constructing nets, which map net-objects to net-objects in general.

Thus the proposed net specification system is founded on the use of a fixed set of combining forms called path forms. A new net can be built from existing ones hierarchically or recursively by means of path forms and simple definitions.

For example, in the net expression

A > (p0)B > [C; D] > (p1){A; (p1); #}

actions A and B is composed through a linker (p0), where certain link condition and termination mode are specified; C and D are carried out in parallel (serializable); the conditions stated in linker (p1) are checked against the

current state to determine whether go back to execute action A, recheck the conditions, or exit.

Thus a generalized Spatial DBMS offers an integrated management to three kinds of objects :

(1) Data/knowledge bases,
(2) A collection of action modules,
(3) A set of operational scheme specifications.

This approach provides a precise formalism for task specification, under which each operational scheme is treated as an integrated object within the system framework, which makes it more reliable to control the data integration and easier to define task-oriented constraints, and can enhance the flexibility and extendability of the system to accommodate applications in multiple domains.

OBJECT-ORIENTED DATA MANAGEMENT

Object-Oriented (O-O) approach is particularly significant to GIS's mixed with a variety of objects in raster, vector, table, ...forms, which are stored as different structures and manipulated in different ways.

From a historical perspective, a number of related notions have been associated with the O-O approach, such as :

- data abstraction and encapsulation,
- object identity independent of (mutable) property values.
- properties inheritance,
- message,
- overloading,
- late binding,
- interactive interfaces with windows, menus and mice.

In the object-oriented programming paradigm, reality is represented in terms of objects as well as their relationships. Each object has an associated set of procedures called methods denoting its dynamic behaviors. The manipulation of objects is made by applying the methods on them, referred to as "sending the objects a message" [Cox 84].

Unlike the usual operator/operand model, which treats operators and operands as if they were independent, in the above message/object approach, an object instance records its type (class) explicitly, which is used to determine the set of legal operations on the objects of this class, and the type dependencies become permanently encapsulated within classes. This concept is coincident with a "save" database design principle : localization [Rid 83].

In the object-oriented semantic modelling paradigm, an important feature is the inheritance network whereby an object can be declared as a specialization of other objects, therefore inheriting their behavioral properties. This feature allows

new classes to be built on top of older, less specialized classes instead of being rewritten from scratch. The inheritance network is declared by means of the infix operator "isa". The clause

object_A isa object_AA & object_BB & ...

implies inheritance to A from its ancestor AA or BB all the methods and constraints. The "isa" relationship is not symmetric, but transitive, in a sense that an "isa" link can be resulted from the transitive composition of others without redundant declarations. When an object has more then one ancestors, we first determine the order of the ancestors, then use a so called "upward-first" search strategy, starting from checking the first available ancestor, observe to see whether it is a root (no further ancestor remained), if not, move upward until either the matching succeeds or backtrack to try the next possible ancestor at the highest possible ancestry level.

In fact we use the O-O approach at two levels :
- First we view raster coverages, vector coverages, relations,...etc as different types of objects, and define a set of legal operations for each of them. Thus a high-level, generic command is interpreted according to its objective variable, and executed by invoking special object-oriented procedures.

- Second, we define objects according to the problem domains, or simply, on tasks, since certain operations are necessary for certain applications, but illegal for others.

In GIS applications we must pay attention to the multiple view of a data set. For example, a base data set for a digital terrain model can be represented as an image (raster object), a map with contour lines (vector object), or a table of data items (table object), each is considered as different type object and associated with different set of operations. Furthermore, each type of objects mentioned above, such as vector coverage type, must be further classified into various subtypes. We see for example, certain operations defined on large scale maps, are meaningless for small scale maps. With a fine O-O management, a spatial database is more distinct and can be used more safely.

VIRTUAL OBJECTS (VO's)

The introduction of VO aims at providing high-level and user-friendly interface, and reducing data redundancy.

VO's contain certain derived data, which may or may not be actually filed and physically stored until needed, but can be directly accessed in terms of usual query commands, thus providing a link between data access and data processing. The instance of a VO is derived from the instances of other (source) objects, and instantiated by carrying out an action

thus representing the resulting data of that action. Note that VO's we discuss here are more general than relational views. As relational calculus only forms a subset of the general function mapping, views only form a subset of VO's.

The declaration of a VO contains structure, source objects, mapping rules or procedures and so on. The source objects can be actual or virtual or a mixture of both, thus a VO may be defined hierarchically.

The instance of a VO has a life time which extends over a single task. Its refreshment may adopt one of the following strategies : (1) when it is accessed after the alteration of its source objects; (2) by demand. During a tast, the cost of recomputating VO's can be reduced by introducing a mechanism called "life flag". The life flag is a property of a VO, indicating whether this object has a valid (live) current instance. For each actual or virtual object a possible list of its "directly dependent virtual objects" (DDVO) is generated automatically by the system from all the VO declarations. For an actual object, any updating automatically triggers a kill operation, turning off the life flag of its direct dependents (if any). For a VO, any kill operation propagates through its own direct dependents (if any). Thus updating an actual object will kill all its virtual dependents hierarchically. Thereby the recomputation of a live VO, which may be directly accessed or indirectly referenced as the source object of others, can be eliminated.

The constraints defined on a VO is interpreted as the post-condition of its mapping action (for validating the instance). There is no update operation defined on VO's.

This concept has played an important role in developing our GIS. Under this approach, results for many operations, such as windowing, feature extracting, merging, can be handled in a unified fashion, as if they were originally stored in the system. Even the multiple views (image, line map,...) to a base data set may be considered as VO's. Thus, by conveying data processing results to users in the form of answering quries, the system usability can be extended considerably, particularly for those who are not computer specialists.

In addition this approach offers the following advantages :

- A VO is a more feasible entity than an action. A VO has a structure definition, a printing format, can, flexibly, either be used as a source object for other VO's, or be involved in any tasks.
- VO provides a natural way to accomplish rule localization.
- Duplicate recomputation of VO's in a task can be reduced.

DYNAMICALLY DISTRIBUTED DATABASE WINDOWS

The spatial data processing requires a significantly diffe-

rent database architecture in buffering and user-interfacing from that developed for conventional business applications, since

- A GIS transaction typically involves much bigger data sets and lasts a much longer time. This feature requires a precise management for the data check-out environment, which allows a team of users to complete a complex transaction involving numerous objects by passing incomplete objects back and forth among them in a controlled manner.

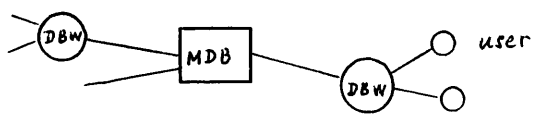
- The modern GIS's are characterized by high degree of function distribution as they are facilitated with versatile intelligent workstations, such as image processing workstations, graphic analysis workstations and so on, all with local processors and buffers for temporarily storing copies of the required database objects. Some of these workstations may be mostly suitable for performing certain types of transactions by means of special hardware and software. For example, an array processor can make an image modification much more effective than the main system can. This feature requires the management of the check-out environment to be distributed yet integrated to the whole system.

- The database objects checked-out are just temporary, swapable copies. This feature requires the system to handle dynamic data distribution, rather than static distribution.

In fact, the data management system in an environment involving multiple check-out copies should be considered as one in between DDB and multicache system. Thus we introduce a concept called dynamically distributed Database Window (DBW).

A DBW resides on a workstation, containing objects copied from the Main Database (MDB), together with the universally quantified integrity constraints on these objects, which provides an extended application-oriented programming environment for the MDB. A user can access both MDB and DBW's. There are communication paths among MDB and distributed DBW's for data check-in/out, and update synchronization.

Thus a DBW is handled as, firstly, a check-out environment holding required copies of database objects; secondly, a semi-independent system supported by a local data manager, where data can be manipulated by multiple users; and finally, a swapable buffer, not for keeping fixed set of data, but for buffering the required data for the current applications.



A DBW is defined by specifying a query to the selected database objects. The definition of a DBW causes the query to be parsed and stored by the DBMS. An OPEN/ADD request can be

made at any workstation, but must be sent by the system to MDB for execution. Managed as a temporary database, a DBW can be queried and updated, while committed updates made to the objects are propagated to the MDB and those DBW's who contain the same objects.

The application program interface includes certain particular language constructs to DEFINE a DBW, to OPEN and CLOSE a DBW, to ADD or DROP objects to or from a DBW, to EXECUTE a transaction (upon which the identification and the timestamp of the transaction are generated), to SUSPEND and RESUME a transaction, to FORK the transaction into a hierarchy of sub-transactions for cooperative task, and to GRANT and REVOKE R and W privileges to specific sub-transactions to use certain objects.

For example, we use the following statement in our GIS to define a DBW named "map" on the workstation "station_a", who contains data for the features "elevation" and "land_use" (physically stored in separate files) of an 1:1000,000 scale digital map "J-47" (in our system "grid" is not a relation, but a generalized logical database object while each coverage is treated as a view of "grid") :

```
DEFINE map ON station_a INCLUDE grid WHERE coverage = "J-47"
          AND feature = "elevation" AND feature = "land_use"
```

(which allows objects grid.dtm.J47 and grid.lnu.J47 be included in DBW map on station_a). We can also delete from the above DBW the data for the feature "land_use" of coverage J-47 as

```
DROP grid FROM map WHERE coverage = "J-47" AND
                         feature = "elevation"
```

To describe the operational behavior of a multi-DBW system we refine some concepts for depicting the system, where all the states mentioned below are stable states, that is, the states detected when there is no unfinished transactions on the involved database objects. We use $X^{\sim} = \{X, \{X1,\ldots,Xn\}\}$ to represent the combination of a datbase object X in MDB and its copies {X1,...,Xn} in DBW's, use D to denote the MDB, use $D^{\sim}$ to denote the combination of MDB with all DBW's.

[DEFINITION]
. A state of a database object X in MDB is referred to as a primary state of the object, denoted by s(X), where the set of all its possible primary states is denoted by S(X).

. A set of primary states of all the objects in MDB, D = {X,...,Y}, is referred to as a primary state of the database, and represented by

$$s(D) = \{s(X),\ldots,s(Y)\} \quad \text{where } X \in MDB,\ldots,Y \in MDB$$

The set of all database primary states is denoted by S(D).

[DEFINITION]
. A state of a database object X in the MDB, together with all its copies in DBW's, {X1,...,Xn}, is referred to as an extended state of the object denoted by

$$s(X^{\sim}) = \{s(X), \{s(X1),...,s(Xn)\}\}$$

If X has no any copy outside then $s(X^{\sim}) = \{s(X), \phi\}$. The set of all extended states of the object is denoted by $S(X^{\sim})$.

. A set of extended states of all the database objects is referred to as an extended state of the whole database denoted by

$$s(D^{\sim}) = \{s(X^{\sim}),...,s(Y^{\sim})\} \quad \text{where } X \in MDB,...,Y \in MDB$$

The set of all possible extended states of the database is denoted by $S(D^{\sim})$.

[DEFINITION] A set of integrity constraints IC(D) is a set of universally quantified predicates on the primary state of database D, denoted as

$$IC(D) : S(D) \longrightarrow \{True, False\}$$

where certain subset of IC is defined on a subset of S(D), such as

$$IC(X) : S(X) \longrightarrow \{True, False\}$$
$$\text{where } IC(X) \in IC(D) \text{ and } X \in D$$

[DEFINITION] A consistency constraint is a mapping CC on the extended state of a database, as

$$CC : S(D^{\sim}) \longrightarrow \{True, False\}$$

or on individual objects, such as

$$CC : S(X^{\sim}) \longrightarrow \{True, False\} \quad \text{where } X \in D$$

[DEFINITION]

. A primary state s(X) of an object X, or s(D) of the database, is said to be integrate iff $IC(X):s(X)\longrightarrow True$, or $IC(D) : s(D) \longrightarrow True$, respectively.

. An extended state $s(X^{\sim})$ of an object X is said to be consistent (that is, $CC : s(X^{\sim}) \longrightarrow True$, or simply $CC(s(X^{\sim}))$) iff

$s(X^{\sim}) = \{s(X), \phi\}$ or

$s(X^{\sim}) = \{s(X), \{s(X1),...,s(Xk)\}$ and $k >= 1$
and $s(X) = s(X1) = ... = s(Xk)$

An extended state $s(D^{\sim})$ of a database is said to be consistent iff

$$\forall X \ (X \in D) \longrightarrow CC(s(X^{\sim}))$$

. A state of an object or a database is said to be valid iff

$IC(X)(s(X))$ and $CC(s(X^{\sim}))$
or $IC(D)(s(D))$ and $CC(s(D^{\sim}))$ respectively

that is, both integrate and consistent.

Based on the above definitions, we clarify below the concept of transaction in a multi-DBW system.

[DEFINITION] A transaction on a database is a mapping from a stable extended database state to another stable extended database state, that is

$$T : S(D^{\sim}) \dashrightarrow S(D^{\sim})$$

[DEFINITION] An execution of a transaction e(T) is correct if the stable state $s(D^{\sim})$ it causes is valid by satisfying $IC(D)(s(D))$ and $CC(s(D^{\sim}))$.

Based on the above definitions the follows are implied :
- A traditional database altering operation can be considered as an action within a transaction. Such an action may cause a temporary inconsistent state, but is undesirable to be followed by an immediate synchronization effort. For the logn-duration, complex and frequently modified GIS transactions at DBW's, benefits offered by this treatment consist in high flexibility and low communication cost.

- There is a data coherence problem among MDB and DBW's, caused by the multiple access paths to a logic database object simultaneously, which is more complex in a multi-DBW system than in a static multicopy DDB.

- A DBW OPEN/ADD or CLOSE/DROP operation must be considered as a transaction. Although it neither changes the primary state of MDB, nor has confliction with another OPEN/ADD or CLOSE/DROP operation, it does cause a transition of the extended state $S^{\sim}$, and may have conflict data set with another transaction.

Thus a transaction involving multiple actions can be modelled as
[DEFINITION] A transaction T is a 5-tuple (A, PA, IC, U, PU) where
- A is a set of actions,
- PA is a partial order on A,
- IC is a set of integrity constraints which must be satisfied by the system states before and after the transaction execution,
- U is a set of post-actions for enforcing the system legality, such as update synchronization, recovery,...etc.
- PU is a set of protocols on U, expressed by a set of rules and algorithms.

In general a protocol is based on a correctness criterion of executions, such as two-phase locking, timestamps ordering,...etc, which guides the acceptance test and the necessary post-execution actions such as update synchronization.

Augmenting all update synchronization approaches to static

distribution, such as global locking, dominant copy synchronization, majority consensus synchronization, multiple protocol synchronization, ...etc, the update synchronization algorithm we developed stresses dynamic distribution, which is described in author's another report [Chen 86b].

Summarizing the above, our model augments existing models by refining the notion of checkout environment and controlling the update synchronization between the main system and its buffers based on the concept of dynamic distribution.

CONCLUSIONS

In this paper we have discussed some issues in designing a spatial DBMS, aiming at getting better compromise of database principles and GIS requirements.

In studying the impacts of logic on databases, our another effort is to develop a rule-based data/task handling approach [Chen 86a]. A formal software specification methodology for information system is explored as well [Chen 85a].

REFERENCES

[Chen 86a] Q. Chen, "A Rule-based Object/task Modelling Approach", Proc. of ACM-SIGMOD 86 International Conference, Washington D. C. 1986, USA.

[Chen 86b] Q. Chen, "The management of Dynamically Distributed Database Windows", Tech. Rep. 1986.

[Chen 85a] Q. Chen, "Extending the Implementation Scheme of Functional Programming System FP for Supporting the Formal Software Development Methodology", Proc. 8th International Conference on Software Engineering, London, 1985.

[Chen 85b] Q. Chen, "Toward A Generalized Data/Action Management : An Approach for Specifying and Implementing Operational Schemes", Proc. 1st Pan Pacific Computer Conference, Melbourne, Australia, Sep. 10-13, 1985.

[Cox 84] B. J. Cox, "Message/Object, An Evolutionary Change", IEEE Trans. On SE, pp. 50-61, Jan. 1984.

[Melk 83a] M. Melkanoff and Q. Chen, "An Experimental Database Which Combines Static and Dynamic Capabilities", Proc. Engineering Design Applications, ACM-SIGMOD'83/Database Week.

[Melk 83b] M. Melkanoff and Q. Chen, "Integrating Action Capabilities into Information Databases", Proc. 2nd International Conference on Databases (ICOD-2), Cambridge, UK, 1983.

[Rid 83] D. Ridjanovic and J. Brodie, "Action and Transaction Skeletons : High level Language Constructs for database Transactions", Proc. ACM-SIGPLAN 83, 1983.