Active Databases as a Paradigm for Enhanced Computing Environments

Matthew Morgenstern

USC Information Sciences Institute
4676 Admiralty Way, Marina del Rey, CA. 90292[1]

Abstract

Active databases emphasize the notion that a body of information is dynamic and should respond intelligently and in non-trivial ways to the user. It provides a paradigm for research and development which combines aspects of both database and artificial intelligence technologies. A prototype system has shown the viability of this approach.

We focus on the following database issues: (1) *Descriptions* are used as semantic templates for associatively accessing and manipulating data objects. (2) *Dynamic views* minimize the typical distinctions between queries and retrievals, and between views and real data, and thereby increase the perceived immediacy of the user interface. (3) *Constraint Equations* are developed as a declarative representation for semantic constraints. The uniform approach they provide for expressing database integrity, consistency, and more general semantics derives its power from the rule-based framework of recent A.I. expert systems. The efficiency of constraint maintenance also is considered. Lastly, (4) The notion of *binding time* of data associations and reference is discussed relative to both the choice of data model and to the method of data access.

1. Introduction

The working environment for computer scientists, managers, and office workers has in common the need to dynamically organize and keep consistent a large complex of interrelated information. Whether the information involves messages, articles, and files, or programs, data structures, and specifications, the need for flexible organization, browsing, and maintenance of consistency is similar.

Active databases provide a paradigm for uniformly addressing the information handling activities that are central to one's computer work environment. The activeness of a database is a behavioral metaphor which involves several dimensions, and emphasizes the dynamics of the user's interactions and the desirability for system intelligence in dealing with consequences and implications of these interactions.

Consider an analogy to the evolution of text editors. In the older line-oriented systems, a set of commands were formulated, sent to the editor, and then the results were viewed separately. By comparison, interactive screen oriented editors (such as Emacs [Stallman79]), have commands which serve more as direct extensions of one's human capabilities. One moves a finger not only to enter a new character, but also to physically move a piece of text. The command is evidenced by its effect, and the feedback is immediate. There is no significant distinction between what one sees and what really is. The level of graphical interaction and responsiveness provided by sophisticated personal workstations afford another example of this kind of immediacy.

Similarly, one would like to reduce or eliminate the distinction in databases between queries and results, and between views and the real database. One should not need to form a complete correct query for a complex question before there is any response; rather, one should be given dynamic feedback as the qualifying conditions are refined.

Views should be dynamically modifiable as to the data which they display. This would enable one to incrementally shift the focus of attention and to browse interactively through the data as is more naturally done in libraries. Related approaches to user interaction have been explored in [Goldstein80], [Robertson81], [Stonebraker82], and [Tou&Williams82].

The derived view should be as real as the underlying data. When the view leaves ambiguity as to how an update should be executed, additional information may be brought to bear.

[1]This research was supported by the Defense Advanced Research Projects Agency (DARPA) contract MDA-903-81-C-0335. The views and conclusions contained in this document are those of the author and should not be interpreted as representing official policies or opinions of DARPA or the U.S. Government. Colleagues involved in the Information Management project are Robert Balzer, Dave Dyer, Neil Goldman, and Robert Neches.

Disambiguating the update may utilize the recent focus of attention [Davidson82], semantic rules, or interactively acquiring more information from the user.

Modelling complex applications such as office systems requires a representation for the semantics of the data. The apparent complexity of many applications is due in part to the complex interactions which arise between a large number of rules or requirements -- rules that individually often are relatively easy to specify. Kunin observes that "the basic structure of the office procedure is often relatively easy to comprehend and describe, given the appropriate set of primitives. The 'complexity' ... is often an artifact ..." [Kunin82].

For example, changing a secretary's office may change her phone number, which may change the backup extension for people on the same project, and also may require a change to the allocation of telephone charges to projects. Each of these changes is simple when stated individually, yet their combined effect appears complex.

The active database can assume responsibility for managing the complex interactions between these rules. Previous approaches to representing database semantics have relied upon additional schema representation primitives and/or procedural representations (see [Hammer&McLeod81] and [Mylopoulos80]). The representation of semantics utilized here is derived from the rule based methodology of expert systems, and emphasizes declarative expression of semantic constraints, with an executable interpretation that provides for maintenance of these constraints.

The work described herein has taken place as part of the Information Management (IM) Project at ISI. This project provides a framework for database and artificial intelligence research. The original high level goals of this project were presented in [Balzer82] and include: (a) providing to the user greater freedom of interaction; (b) managing a broad range of information in a uniform manner, including messages, program specifications, and code modules; (c) coordinating interdependent data to maintain integrity and consistency; and (d) providing a high level means of expressing frequently performed activities which are to be automated so as to extend the functionality of the system.

In the subsequent sections we present the concept of *Descriptions* as the basis for data access and for the creation of new entries. Descriptions also provide the defining criteria for *Dynamic Views*, which facilitate browsing and interaction with the system. The concept of *Constraint Equations* is developed as our declarative representation of semantics, which can be used to augment typical database schemata. Efficient maintenance of these constraints also is considered. The relevance of binding times to methods of data access and to the choice of data model are highlighted. We conclude with a review of the implementation features for the current IM prototype system.

2. Descriptions and Browsing

Descriptions are expressed in terms of the data model, which for our purposes can be classified as an entity-relationship data model. The entities or objects are classified into a type lattice that supports inheritance of attribute and relationship definitions from one or more parent object types. The type lattice is dynamically modifiable. Objects may have multi-valued attributes and may participate in n-ary relationships involving other objects.

Descriptions provide a semantically based form of associative access. A simple Description refers to a type of object and provides a skeleton of the attribute and relationship names for that object. (A Description also may refer to a relation (simple or derived) and its roles.) Specific values and restrictions may be entered to characterize a subset of object instances. For example, a Description may be formed for an EMPLOYEE object type, and may specify the IM Project for the Works-on attribute.

This may be represented as:

```
[ EMPLOYEE Works-on:IM ],
```

where the square brackets denote a Description, and the first entry is the type of the object, followed by one (or more) Attribute-name:Value pair. In the system, a Description usually is built interactively.

A compound Description is a set of simple Descriptions which participate in common relationships. To locate Employees who are Researchers working on a Project funded by DARPA, one forms a Description of Project having the Funded attribute value of DARPA, and relates this Description to an Employee Description whose job title is Researcher. To express this syntactically, the Employee Description below has for its Works-on attribute an embedded Description for Project:

```
[ EMPLOYEE  JobTitle:Researcher
            Works-on:[PROJECT  Funded:DARPA] ]
```

Alternatively, two or more Descriptions may be related by linking-variables so as to express the fact that several different attributes have the same value. The use of linking-variables has additional power in that more than one association can be established. In the example below, we are interested in Employees who are Researchers and who both Work-on and are the Principal for a Project Funded by DARPA. *PROJ* is the linking variable which denotes a value for both the Works-on and the Principal attributes (in the first Description), as well as an instance of a PROJECT object (in the second Description). If there are multiple values for *PROJ*, then there are multiple ways of satisfying the following compound Description:

```
[ EMPLOYEE  JobTitle:Researcher
        Works-on: *PROJ*   Principal:*PROJ* ]
[ PROJECT:*PROJ*   Funded:DARPA ]
```

Descriptions provide frameworks or templates for objects into which values and relationships to other objects may be entered. This is analogous to Query By Example's use of relation table

skeletons for two dimensional entry of relational queries [Zloof77, Zloof82]. Since our objects are defined relative to the application domain, the object Description skeletons used to formulate interactions with the database reflect the semantics of the application rather than some partitioning into underlying component relations.

A compound Description which relates several simple Descriptions may serve to widen one's focus of attention to include other related objects, or to provide additional qualifications in the context so as to filter the set of object instances which satisfy the Description. If parts of the compound Description are designated as not being visible, then they serve only as a filtering context for the objects and attributes which are visible. Descriptions are utilized for Descriptive Reference, and are the basis for the notion of Dynamic Views, as described below.

2.1. Descriptive Reference

Descriptive reference emphasizes the use of context in selecting and qualifying the set of objects of interest, in contrast to name based and access path based modes of reference common to some traditional database systems. Partial specification of related information in the Descriptions serves to select the objects which are desired for access.

The objects in our database can include rather general types of information, including electronic messages, program specifications, and subroutine code. Thus Descriptive reference provides a possible tool for categorizing messages, forming a mailing list based on common interests, and even for function selection and invocation based upon types of inputs, outputs, and behavioral features associated with the stored functions.

Descriptive reference provides for not only explicit retrieval, but also retrieval by analogy, object creation, and modification. For example, one might wish to use an abstraction or generalization of one object to find similar objects. This capability is realized in our system by being able to form a Description that matches a given object, and then using this Description, with modifications, to select objects of interest. As the user explicitly removes restricting conditions, such as specific attribute values, similar objects that satisfy the more general Description are brought automatically into focus.

Objects may be created based upon similarities to existing objects, utilizing a concise differential description of the new object relative to an existing object. Specifically, a Description of that existing object is formed, and the differences for the new object are expressed by changes to this Description. The new object is created by instantiating this modified Description.

Instantiation of a Description is one example of the act of *anchoring* a Description to a specific object -- here a new object. A Description is considered unanchored when it is used as criteria for selection of the objects in focus. Anchoring the Description serves to freeze [Balzer83] the binding to the object. Then subsequent modifications affect this object directly.

2.2. Dynamic Views and Browsing

Interactions with the database begin by establishing a *focus of attention* -- that is, locating specific objects of interest. For example, a Description selects and brings into the user's focus of attention those Employees which satisfy certain criteria. However, a Description goes beyond the notion of a query and retrieval. It can define a *Dynamic View* of the database -- a modifiable window on the real data. The Dynamic View has a time-varying extension consisting of those object instances which satisfy the Description.

Changes to this Description are automatically reflected in the satisfying set of object instances which are in focus -- without explicit steps of retrieval. Updates to visible objects will be shown as they occur, and modified objects will be retested against the current view Description to determine if they should appear in the current focus.

This spontaneousness of effect reduces the distinction between queries and retrievals, and provides an interface which is more active in response to the user. These capabilities are implemented by coordinating the Description with the set of object instances which satisfy it (using the triggering capabilities described below). Some or all of these objects in a Dynamic View may be displayed in summary form, and the set is scrollable if the number of instances is large. Instances may be selected by a pointing gesture (via a mouse or screen commands) and expanded to see a larger set of attributes and relationships to other objects.

One also may browse to related objects and information by following relationships that emanate from one's focus and extend beyond it. Doing so augments the focus. Thus if while viewing an Employee, one points to the Lives-at relationship, part of which extends outside the current field of view, then the relevant instance(s) of the Residence object will be brought into the view.

The schema itself may be interactively inspected and augmented. When one is focused on the definition for the Employee object type, one can browse to the definition for other related object types, such as the definition of the Office object. Or one can focus on selected Employee instances. Similarly, from an Employee instance one can move to the definition, or to other object instances.

Thus the schema can be used not only for its definitional properties, but also for the pathways it provides between classes of objects. The schema becomes, in some sense, an additional dimension along which one's attention may be directed.

In summary, Descriptions provide the basis for Dynamic Views

and for Descriptive Reference. Compound Descriptions define an interconnected set of information which is to be kept in view. Descriptions can be specified interactively and/or derived by analogy to an existing object. Both simple data and more complex objects such as program specifications are accessible via Descriptions. In addition to retrieval, Descriptive Reference also provides for modification and creation of objects and relationships.

3. Expressing and Maintaining Consistency and Semantics

The integrity and consistency of a database require that a variety of implicit and explicit constraints be maintained among the data. Examples of such constraints include whether a dependency is functional or multi-valued, whether the existence of one object requires the existence of another object (e.g. can an employee exist without a representation for the company he works for), whether an attribute or relationship is mandatory or optional, and whether an attribute is of limited cardinality or arbitrary in number.

A large class of consistency constraints arise from semantic interdependence between several relationships. This may take the form of equivalences of data that are related by a chain of associations; or it may take the form of a computational derivation, such as an aggregate value, a conditional value, or a one-to-one transformation of values (e.g. Cartesian versus polar coordinates or Department Number versus Department Name). Consistency constraints need not always provide a complete derivation of the allowed or desired data, but may constrain the allowed set of values (eg. the State constrains the U.S. Zip codes that are possible for that address).

Various data models incorporate certain constraints (eg. existence dependency, uniqueness, etc.) while other constraints are more difficult, or impossible, to represent in the schema. Data models differ with respect to their coverage of these constraints and the defaults they assume. Typically, these constraints are embedded in the declaration of the keys of a relation, or in the parent and child segments of a one to many relationship, for example. By making these constraints and dependencies explicit and separable, the perceived differences between these data models may be reduced [Morgenstern81]. Furthermore, the ability to declaratively represent additional semantic constraints would be a natural extension to existing database schema declarations.

Perhaps the earliest approach for augmenting schema descriptions with additional semantics were the database procedures of the CODASYL network database [Wiederhold77], which provide a means of invoking procedures to derive data or perform limited additional actions. More robust approaches include providing additional primitives in the data model representation [Hammer&McLeod81], utilizing semantic nets and/or attached procedures, or a combination of these (see the TAXIS system in [Mylopoulos80]). Recent extensions to the INGRES database system use a QUEL-like syntax for expressing rules [Stonebaker83]. A rule is selected when it syntactically matches a user query, which is then modified by the rule.

The alternative utilized here draws upon production rules (condition-action rules) which form the cornerstone of recent expert systems that have been developed in the Artificial Intelligence community [Buchanan82]. Also relevant are studies of constraint-based systems, including [Borning79] and [Goldstein80].

Condition-action rules have the advantage of being modular and easy to specify, yet a set of such rules can express complex knowledge and actions. For example, a consistency constraint expressed as a condition-action rule would state the change or combination of changes to the data base that serve as the condition for activating the rule. And it would state the action to be taken when that condition is satisfied -- typically an expression of how to reinstate consistency given this change to the data. Other forms of action might be to disallow the change, provide information to the user, or invoke a more general procedure to execute an arbitrary action.

A simple example is the triggering of a rule for maintaining aggregate data based upon a change, say, to an employee's salary. This change triggers the rule and binds it to the particular employee instance. The action part of the consistency rule utilizes this employee instance to locate the employee's department and updates the associated average salary attribute accordingly.

3.1. Constraint Equations

A *Constraint Equation* (CE) is a declaration of an *invariant* relationship that is to be maintained among specified data objects -- in contrast to a more procedural statement of what actions to take under what conditions. A CE provides a declarative representation for a set of related condition-action rules. As an analogy, the algebraic equation X = Y + Z declares an equivalence between its two sides. If this is to be treated as a constraint which is to be maintained by the system, then there is an executable interpretation which may be thought of as two condition-action rules: (1) if Y and/or Z change, then revise the value of X accordingly, and (2) if X changes, obtain more information so as to select between the alternatives of disallowing the change, revising Y, or Z, or both.

The language of *Constraint Equations* provides a declarative representation for commonly occurring kinds of semantic relationships. For example, the above semantics of a Department's Average Salary can be expressed as

```
DEPT.AVG-SALARY  ==
                    AVERAGE [ DEPT.EMPLOYEE.SALARY ]
```

The brackets denote a Description, which here contains a *path expression*. This path expression describes the sequence of associations from a Department (same binding as on the left side of the CE) to its Employees and then to the set of all their

Salaries, which is the result produced by the Description -- thus this Description is for objects of type Salary. This path expression assumes that there is a single relation between Department and Employee, and a single relation between Employee and Salary. When this is not the case, the relation name must be given to eliminate ambiguity. In the above path expression, the dot may be thought of as a form of ellipsis for the relation name.

Note that if an Employee had more than one Salary then all of them would be included here. This constraint is to be maintained for all satisfying Department instances. In this example, an attempt to change the Average Salary directly would be ambiguous as to its effect on each Salary, and thus would be disallowed, unless further information is given in the rule or interactively by the user.

A semantic consistency rule which does not involve aggregate data is the rule which states that an employee's backup phone for messages is the phone of that employee's project's secretary. It could be expressed by several simple production rules, or by one rule whose condition part is a disjunction of tests for each of the changes which may arise: an employee moves to a new project, a new secretary takes over for the project, or the telephone number of the secretary is changed (perhaps as the consequence of other actions which are reflected in the database, such as the secretary moving to a new office).

This semantic relationship may be concisely and naturally expressed as a Constraint Equation which states that an Employee's Backup-Phone is equivalent to following the path expression (sequence of associations between objects) from that Employee to his/her Project, to the associated Secretary's Phone:

```
EMPLOYEE.BACKUP-PHONE  ==
                EMPLOYEE.PROJECT.SECRETARY.PHONE
```

Though this Constraint Equation is declarative in nature, it has an executable interpretation similar to that of the condition-action rules described above. In particular, a change to any of the three associations described on the right would serve as the triggering condition. The association which changed binds part of the Description represented by the right hand side. Each possible way of binding the remainder of this Description identifies an Employee instance whose Backup phone is to be revised.

For example, changing the Project's Secretary would bind the Project and the (new) Secretary. For each Employee on that Project, the Backup-Phone due to the old Secretary is replaced by the new Secretary's Phone. It is the rightmost components of the two sides of the CE that are being equated. If there are multiple Secretaries, then the default meaning here is that the set of Backup-phones of an Employee is to be the same as the set of Phones for all the Secretaries associated with that Employee.

For the executable interpretation of the above CE, we have treated the right side as independent and the left side as dependent, thus far. Changing this directionality of interpretation is valid, but there is not yet sufficient information to make this update unambiguously. That is, given a change to the association between an Employee instance and his/her Backup-Phone, which one (or more) of the three associations on the right should we change, or do we disallow the initial update to the Backup-phone?

This is a special case of the view update problem [Dayal78] in that the update is not functionally determined. Here there are a small number of alternatives which are a consequence of the schema and the form of the constraint. In some cases, one might say that one of these alternatives is far more natural than the others, and should be taken as the default unless further information is explicitly provided. Or perhaps the context of the query can help to rule out some of these alternatives [Davidson82].

This ambiguity can be resolved, when there is a specific intent, by separating the right side of this CE into two parts by denoting a "weak bond" association which is subject to revision. (The two parts on either side of this weak bond each can be considered as a tightly bound cluster of information, and could be bracketed as sub-Descriptions to emphasize this.) It is this weak association which is to be revised when that side is treated as the dependent side.

The above Constraint Equation is rewritten below with the relationship between Project and Secretary denoted by ".!" to indicate it as the weak association on the right side:

```
EMPLOYEE .! BACKUP-PHONE  ==
                EMPLOYEE.PROJECT .! SECRETARY.PHONE
```

Thus directly updating an Employee's Backup-Phone means that the association from the Employee's Project to the Secretary is to be revised. The Secretaries to remove and to add for this update are determined by the old and new values for the Secretary's Phone. In the above CE, the weak association on the left side is also made explicit. In general when there is a single association on a side which is being treated as dependent, it is assumed to be a weak association -- unless indicated otherwise.

If the dependent side is enclosed in one set of brackets to denote a single tightly bound cluster of information, then there is no weak association, and the change which triggered the independent side should be disallowed if it violates the invariant. Multiple weak associations on the dependent side mean that the ambiguity is limited to just these weak associations, but the other relationships are not candidates. The user could be queried to resolve the remaining ambiguity.

Constraint Equations represent an important part of the Structural Semantics of the application, in that they can augment common data base schemata with additional relationships and constraints expressed declaratively. It is

anticipated that a large portion of integrity, consistency, and other commonly occurring constraints can be expressed in this manner.

Using CE's has several desirable properties: the semantics of the application are represented modularly and in a form which is indicative of their meaning. They are specified non-procedurally and without concern over order. The modularity is two-fold: additional Constraint Equations can be added incrementally, and each such equation is specified with respect to a local context of relevant objects and relationships.

Furthermore, the declarative form for Constraint Equations makes them amenable to analysis and symbolic transformations (under development) which preserve the intended semantic invariance but which give rise to alternative interpretations for domain modelling and for implementation. (For a discussion of compatibility between different data models see [Morgenstern81].) Since the CEs represent logical assertions, they can be used to reason about the application domain and to prove other assertions.

The system manages the potentially complex set of interactions which can occur as one constraint creates other database changes, activating other constraints, and triggering a succession of related actions. The active database, then, can perform rather complex responses to a user's interaction, yet each semantic constraint and action can be specified separately with rather localized information.

3.2. Constraint Maintenance

Straightforward maintenance of the Constraint Equations utilizes a triggering mechanism that invokes specific procedures, called demons, when database changes occur (described below). As the number of constraints increases, the efficiency of this maintenance becomes important, and the range of alternative strategies needs to be explored. One spectrum for constraint maintenance may be characterized by *when* the constraint is enforced -- that is, the *binding time* for the dependent side of the Constraint Equation.

Most obviously, the maintenance of the constraint may be done immediately when the original change occurs -- the earliest time of binding (*immediate propagation*). Alternatively, maintenance can be delayed until the dependent attribute is retrieved -- a late time of binding (*propagation when used*). When timeliness of the data is not critical to the user, maintenance can be delayed even further.

Along the spectrum between these alternatives is an intermediate strategy of delayed constraint propagation and maintenance -- in which the time of propagation and binding is determined based upon other criteria. This option has been referred to as opportunistic evaluation -- both in the sense of doing the work of constraint propagation when the computer is idle [Balzer82], and in the sense of using priority ranking of the constraints to select the order in which they should be considered for propagation.

When maintenance of the constraint is delayed until use, there is the additional option of whether to store the dependent value or association which has been derived. If not stored, then we have the common approach for derived data, in which the derivation is done anew for each reference -- a viable choice if updates are expected more frequently than retrievals.

When maintenance of the constraint is done opportunistically (delayed propagation), the occurrence of a change must alert all dependent constraints and users that any previously stored data may be dirty -- this can be a difficult problem if the chain of dependencies is long. When recomputation is done, the resultant value will be remembered for future use -- in effect a *memo* is made of this result.[2] If old data can be tolerated by the application, then additional flexibility is possible for delaying the propagation of constraints further based upon relative priorities. Both the semantics of the application and operational statistics -- frequency of use, selectivity, and sparseness -- can be used to improve the efficiency of constraint propagation and maintenance.

It is interesting to relate these distinctions to the notions of *forward chaining* and *backward chaining*, which arise in A.I. inference work. The latter characterize the direction with which a chain of associations or logical inferences is followed. Forward chaining starts from the updated data, following the chain of associations or inferences forward to the consequences. Backward chaining characterizes the process of following the chain of associations from the dependent consequent object (or goal) which is being accessed, back to the source data to determine if changes might affect this dependent object (and rederiving information as needed). Thus *immediate propagation* of constraints is in effect *forward chaining*, while (re)calculation of derived data upon use may be thought of as a form of *backward chaining*.

The efficiency of constraint maintenance also is affected by the efficiency of access to objects, attributes, and relationships. Software caching of relevant indexes, and of object instances which are retrieved from secondary storage, can be helpful. Very promising are several forms of compile time aggregation which can avoid duplicated access to the same data. For example, several separately specified Constraint Equations can be aggregated into one demon or procedure based upon either (a) a common triggering pattern -- *parallel aggregation*, or (b) based upon a chain of constraints such that one constraint takes an action which serves as a trigger for another constraint -- *chained aggregation*. More elaborate compile time analysis could determine which user programs have the potential of triggering specific constraints, and then expanding the code for these constraints in the user's programs.

[2] If there is a valid memo value available when access is attempted then it is used, otherwise recalculation occurs. In either case the results are the same, but the efficiency may differ. The set of memo values may be thought of as a *memo function*.

The richer the semantic model and the more dynamic and responsive we wish a system to be, the more important is efficiency in the face of increased complexity. The alternatives just discussed include the relative time at which constraint propagation is executed, the direction of such propagation, maintenance of newly derived results, caching of information when cost of re-access and likelihood of re-use are high, and aggregation of constraints to reduce net overhead. In the long term, the system should be taking an active role in choosing among these alternatives.

4. Binding Time of an Association

The time varying nature of the set of objects which satisfy a fixed Description highlights an already existing phenomenon: the time at which an association is evaluated (the Binding Time) can alter the meaning of that association. For example, if one is interested in the set of students taking course CS-503, does one mean the snapshot of those students registered at some fixed point in time, or does one mean the possibly time varying set of students satisfying the predicate "students registered for CS-503."

Typically database query languages retrieve a copy of data which satisfies the query, thereby providing only a snapshot of the data. However, many applications are better served by viewing the dynamic set of information which satisfies the predicate. Examples include the set of overdrawn accounts or the set of ships entering a critical zone. Our use of active Descriptions provides this dynamic interpretation of a query. Snapshots of data become an option when timeliness of information is of lower priority, rather than snapshots being the only option.

The notion of binding time of an association also applies to assumptions in the data model, and sometimes to the implementation chosen for the database. (An independent discussion of binding in information systems may be found in [Wiederhold81].) Typically, relational databases consider each normalized relation as being stored separately. Combinations of information are created when requested by taking joins of the relevant relations. A tuple resulting from a join represents an *association by description* -- that is, a dynamic association based upon the join attributes of the component tuples satisfying the join description.

On the other hand, object-based data models typically maintain explicit *associations by linked reference* between objects utilizing links (either pointers or explicit relationships between the object id's). These links are established when the association is first explicitly created, rather than when use of data values indicates that this association is relevant. These *associations by linked reference* are comparatively static, in that they require explicit revision when the data no longer justifies the association. (This distinction is minimized in those cases where stored data is not redundant with the association, and thus the association itself is essential information.)

Data models in both the Database world and the Artificial Intelligence world, tend to explain their semantics partly in terms of their implementation, rather than solely in terms of functionality. This tendency in A.I. systems is discussed in [Brachman83]. The functionality of a representation would make explicit the binding time of associations regardless of whether these associations are maintained symbolically by description or physically by pointers. Another example of this tendency to describe semantics partly in terms of implementation is the common A.I. represention of inheritance in terms of a tree or lattice of nodes and pointers, rather than in terms of a functional description of which characteristics are to be inherited along possibly different kinds of inheritance relationships.

The time at which an association or derived relationship is evaluated has an analogy to the way programming languages establish the association between formal and actual parameters for subroutine invocation. Call by Reference utilizes an address or pointer which is bound at invocation time to a particular location -- compare *association by linked reference* between objects as done here. This parameter binding occurs before the use of the variable. A more delayed form of binding is Call by Name -- which utilizes the name or description provided as the actual parameter to determine, at each use, the location and value to be used for that parameter -- compare *association by description* between objects as done here.

The spectrum of possible binding times highlights one of the dimensions along which data model representations may differ, and it also highlights the degrees of immediacy possible at the user interface.

5. Prototype Development

In this section, the current implementation status of the IM prototype is reviewed. As noted earlier, the data model is based upon typed objects and their relationships. An object instance is represented by an internal identifier and a set of relation tuples which have this identifier in common. These relations are normalized and are stored in an inverted index database. Typically objects are accessed associatively rather than by name. Currently all attributes are indexed, though the option exists to reduce update costs by selective indexing. Many to many relationships between objects are naturally provided by this system.

The database supports triggering of demons upon insertion, deletion, and/or update to the database relations which comprise an object. A demon consists of a predicate filter and arbitrary action code. When modification occurs to a relevant relation, the demon is awakened, its predicate is applied, and if it is satisfied the demon is considered activated. Since multiple demons can be activate, they are placed on a priority scheduling queue for execution.

Demons are specified modularly with respect to the object types

involved. Since the actions of a demon may cause changes to the database, an interrelated set of demon activations may result. A collection of user actions and consequent demon activations may be treated as a transaction which must succeed in its entirety before it becomes permanent.

This demon mechanism provides the implementation basis for maintenance of constraints and semantic rules. It also can initiate more general database coordination such as calculating next month's sale's quota given this month's actual sales. And it can invoke arbitrary procedures for automation of actions which can affect the outside world (eg. printing a report, sending a message, etc).

The inverted index relational database facility and handling of demons are provided by the AP3 software system developed by Neil Goldman [Goldman82]. The IM prototype runs on the DEC-20, VAX Unix, and Symbolics 3600 systems.

The application areas which have been addressed or are under development include employee records; a personalized database for notes, mail, and articles; and a database of program specifications and code segments. Rather than treating each such service separately, IM is providing a uniform interface to an integrated set of capabilities.

6. Conclusion

Interaction with the database utilizes *Descriptions* to express the selection criteria for queries, updates, and the creation of new entries, as well as to form *Dynamic Views* for browsing through the database. *Constraint Equations* are a concise declarative representation for modularly expressing semantic constraints, including integrity and consistency. These Constraint Equations have an executable interpretation that enforces the domain specific semantics as the information base changes.

The concept of *active databases* emphasizes the notion that what the user sees instantaneously reflects what exists, including changes and consequences of those changes. Commands are evidenced by the actions that they accomplish rather than by a separate syntactic expression of the commands and later inspection of the results. That the system automatically maintains the consistency and semantics of the data furthers the reality that the database is an active body of information that responds intelligently to the user.

Acknowledgments

The work described herein is an outgrowth of the Information Management (IM) system at ISI. Special recognition goes to Robert Balzer, Dave Dyer, Neil Goldman, and Robert Neches, who, together with the author, have developed the prototype IM system. Michael Fehling also contributed to earlier work on this project.

In addition to the above people, I also wish to thank Jeff Barnett, Don Cohen, Jack Mostow, Bill Swartout, and Dave Wile for their comments.

References

[Balzer82] R. Balzer, D. Dyer, M. Fehling, & S. Saunders, Specification Based Computing Environments, Proc. 8th Very Large Data Base Conf., Mexico City, Sept. 1982, pp.273-279.

[Balzer83] Robert Balzer, private communication, March 1983.

[Borning79] Alan Borning, Thinglab - A Constraint-Oriented Simulation Laboratory, Stanford Univ. report STAN-CS-79-746, July 1979, Ph.D. thesis.

[Brachman83] Ronald J. Brachman, R.E. Fikes, & H.J. Levesque, KRYPTON: A Functional Approach to Knowledge Representation, Feb 1983 Draft, submitted to IEEE Computer, 18pp.

[Buchanan82] Bruce G. Buchanan, & Richard O. Duda, Principles Of Rule-Based Expert Systems, Dept. of Computer Science, Stanford Univ., Aug. 1982. Report No. STAN-CS-82-926, also as No. HPP-82-14. (To appear in M. Yovits (ed.) Advances in Computers, vol.22, Academic Press, New York.)

[Dayal78] U. Dayal & P.A. Bernstein, On The Updatability Of Relational Views, Proc. 4th Very Large Data Base Conf. West Berlin, Sept. 1978.

[Davidson82] Jim Davidson and S. Jerrold Kaplan, Natural Language Access To Databases: Interpreting Update Requests, Stanford Univ. Computer Science Dept., 1982 (submitted to the Amer. Jour. of Computational Linguistics).

[Hammer&McLeod81] Michael Hammer, & Dennis McLeod, Database Description with SDM: A Semantic Database Model, ACM Trans. on Database Syst., v.6, no.3, Sept 1981, pp.351-386.

[Goldman82] Neil M. Goldman, AP3 Reference Manual, June 1982, USC Information Sciences Institute, Marina del Rey, CA.

[Goldstein80] I.P. Goldstein & D.G. Bobrow, Descriptions for a Programming Environment, Proc. First Annual Conf. Nat'l Assn for A.I. (AAAI-80), Stanford, CA, August 1980.

[Kunin82] Jay S. Kunin, Analysis and Specification of Office Procedures, M.I.T. LCS TR-275, Ph.D. thesis 1982, p.39.

[Morgenstern81] Matthew Morgenstern, A Unifying Approach for Conceptual Schema to Support Multiple Data Models, Proc. 2nd Int'l Conf on Entity-Relationship Approach, Washington, D.C., October 1981, pp.281-299.

[Mylopoulos80] John Mylopoulos, Philip .A. Bernstein, & Harry K.T. Wong, A Language Facility for Designing Database-Intensive Applications, ACM Trans. on Database Syst., v.5, no.2, June 1980, pp.185-207.

[Robertson81] G.D. Robertson, McCracken, and A. Newell, The ZOG Approach to Man-Machine Communication, Int. J. Man-Machine Studies, 14, pp.461-488, 1981.

[Stallman79] Richard M. Stallman, EMACS: The Extensible, Customizable, Self-Documenting Display Editor, Massachusetts Institute of Technology, A.I. Lab Memo 519, June 22, 1979.

[Stonebraker82] Michael Stonebraker and Joseph Kalash, TIMBER: A Sophisticated Relation Browser, Proc. Eighth Int'l Conf on Very Large Data Bases, Mexico City, Sept. 1982, pp.1-10.

[Stonebraker83] Michael Stonebraker, et.al., Implementation of Rules in Relational Data Base Systems, Univ. of California, Berkeley, CA., Electronics Research Lab, Memo No. UCB/ERL 83/10, June 13, 1983, 10pp.

[Tou&Williams82] F.N. Tou, M.D. Williams, R. Fikes, A. Henderson, & T. Malone, RABBIT: An Intelligent Database Assistant, Proc. of the AAAI 1982 Nat'l Conf. on Artificial Intelligence, Aug 1982, pp.314-18.

[Wiederhold77] Gio Wiederhold, Database Design, McGraw Hill, 1977.

[Wiederhold81] Gio Wiederhold, Binding in Information Processing, Department of Computer Science, Stanford University, May 1981, Report No. STAN-CS-81-851.

[Zloof77] M.M. Zloof, Query-By-Example: A Data Base Language, IBM Systems Jour., v.16, no.4, 1977, pp.324-343.

[Zloof82] M.M. Zloof, Office by-Example: A Business Language that Unifies Data and Word Processing and Electronic Mail, IBM Systems Jour., 21,3, 1982