Association for Information Systems
AIS Electronic Library (AISeL)

ICIS 1989 Proceedings

International Conference on Information Systems (ICIS)

1989

CONSTRAINT BASED ANALYSIS OF DATABASE UPDATE PROPAGATION

Joan Peckham
University of Connecticut

Fred Maryanski
University of Connecticut

Fred Maryanski
University of Connecticut

Heidi Chapman
University of Connecticut

Steven A. Demurjian
University of Connecticut

CONSTRAINT BASED ANALYSIS OF DATABASE UPDATE PROPAGATION

Joan Peckham
Fred Maryanski
George Beshers
Heidi Chapman
Steven A. Demurjian
University of Connecticut

ABSTRACT

Semantic and object-oriented data models provide convenient constructs for the specification of objects, relationships, and operations. The vehicle of representation is a collection of abstractions which parallel the means by which humans prefer to organize complex enterprises. These constructs inherently permit focus on one object, relationship, or operation at a time. Propagation, as a semantic construct, provides the extension of existing modeling capabilities by providing a mechanism for the specification of the update semantics between database objects. Through the analysis of constraints and the propagated actions necessary to maintain them, we attempt to do the following: 1) incorporate additional semantics into the database schema in the form of database propagation rules, 2) in the context of constraints and propagation rules, provide a model independent paradigm for determining if schemata are correct, and 3) provide a vehicle for the explicit specification of update actions during database schema design.

1. INTRODUCTION

Considerable attention has been given in the past decade to semantic and object-oriented databases. Semantic databases (Hull and King 1987; Peckham and Maryanski 1988) grew out of a desire to permit the database designer to rise above record-level modeling, to model objects and their relationships as they are perceived by the end user. Active or dynamic databases, an extension of static semantic databases, were investigated (King and McLeod 1984; Mylopoulos, Bernstein, and Wong 1980; Brodie and Ridjanovic 1984) as a means of specifying the operations of the database. Object-oriented databases (Dittrich 1986) use a programming-language paradigm to provide an integral technique for the design of schema objects and associated operations.

Also during this period, database applications with increasingly complex semantics have appeared. For example, CAD/CAM, software-engineering, and management information systems require databases which are more complex than traditional applications such as automatic banking systems. For the next decade, attention to the enrichment of database design tools will be necessary for the correct development of more complicated schemata.

In this paper, we focus our attention on the analysis of the objects, constraints, and update actions that are specified during the design of semantic databases. An attempt is made to describe these database semantics so that conflicts and implicit and/or redundant semantics can easily be detected and presented to the designer. This representation also serves as a descriptor of the classes of conflicts which can be detected at design time.

The term "semantics" is assumed to represent the conditions that the designer, when using a particular model, places upon database constructs in order to maintain data correctness and consistency. This includes the design time specification of valid database states and rules by which the active database must execute. These specifications may take many forms, including attribute-range constraints and relationships between database objects.

For each semantic specification, there must be associated actions known as *propagated actions*. For example, stating that an attribute value must be within a specified range implies that a compensating action must be taken should this condition arise. Similarly, an insertion/deletion rule specified for a relationship between two database objects has associated with it a group of compensating actions which uphold that rule. These actions may propagate further actions, whose results must be guaranteed to uphold the semantics of the database. A database design aid must assist the designer in the correct specification of these semantics and help the designer to identify and eliminate any potential inconsistencies.

This research represents an attempt to uniformly capture a wide range of database *semantics*. Incorporating more semantic knowledge into the schema analysis process will encourage better specification of the application environment by the designer. This will limit opportunities for coding errors. This approach may have the potential to

improve the management and coordination of large projects and to reduce software maintenance costs.

2. RELATED RESEARCH

The distinguishing characteristics of this research from prior work in the areas of update propagation, constraint analysis, and dynamic (active) databases appear below.

Research on the dynamic aspects of object-oriented and semantic databases (King and McLeod 1984; Mylopoulos, Bernstein and Wong 1980; Brodie and Ridjanovic 1984) has emphasized the design of operations and transactions. Strategies for the design of operations are important to this research only in the sense that they generate valid and correct schemata for the proper update behavior of the database.

Wilson (1980) and Carlson and Adarsh (1985) have investigated the use of triggers or filters which are initiated at run time to decide if an update can be performed without violating the integrity of the database. In our investigation we attempt to maximize the amount of checking to be performed at database design time. This does not mean that the techniques developed here replace run time checking; rather we intend to complement it.

The work of this paper is most similar to that which has been done by Urban and Delcambre (Urban 1987; Urban and Delcambre 1989). In their work, semantics in the form of constraints are captured at design time using first-order logic. This representation is then employed to aid the designer in the correct definition of perspectives or views. These perspectives are then used for the specification of operations and exceptions on the database. The differences between their work and the work proposed here are:

(1) Urban and Delcambre have characterized database constructs through a synthesis of the predominant features of recent semantic data models. We hope to identify the underlying structures which are used as constructors of the relationships and constraints offered by current models. It is felt that this will provide a tool capable of integration with a broader class of semantic data models.

(2) While Urban and Delcambre assume that the static modeling design process produces no conflicts in definition, we assume conflicts might occur. We attempt to characterize the types of conflicts and redundancies in definition which might lead to incoherent design before active semantics are specified.

Abiteboul and Hull (1987) have examined database update propagation in the context of their formal semantic model, IFO. Three fundamentally different relationship types are utilized to represent the constructors commonly found in current semantic models. Formal descriptions of these relationship types and the means by which they may be combined to form valid IFO database schemata are given by Abiteboul and Hull. They investigated the effect of updates upon IFO schemata and provided definitions of "permissible" updates. Our work differs from that of Abiteboul and Hull in the following ways.

(1) Instead of examining a representative selection of relationship types, we wish to consider the fundamental constructors of the types used in current models. We feel this approach is capable of providing a more meaningful analysis and comparison of existing types and will permit easy extension to newly defined database constructs.

(2) We wish to more explicitly define the update semantics associated with database schemata constructors.

(3) IFO utilizes a set of types and design rules which result in cycle free schemata. This approach assures valid schemata and prevents possible update conflicts, but may needlessly hamper the designer when attempting to create a true model of the enterprise in question. For example, an application needing cyclic, non-hierarchical representation would not be specifiable using IFO. In this work, we wish to relax the IFO design rules to consider the possibility of cyclic object definitions.

We acknowledge that there are always tradeoffs between offering designers total freedom to create overly complex and incomprehensible models and forcing certain modeling constructs and methodologies upon them. Here we eschew the practice of selecting specific modeling constructs and philosophies to accomplish the following:

(1) Investigate the problem of update propagation in databases, independent of modeling philosophy.

(2) Provide an analysis technique which is applicable to a wide spectrum of modeling environments.

(3) Provide a careful characterization of constraint types and the propagated actions which follow from them.

(4) Prepare the groundwork for later investigation into the possibility of including semantically consistent relationships between objects.

Thus, we investigate constraint types and their interrelationships with database actions. The next section describes the paradigm used to classify and represent these constraints and compensating actions. Section 4 shows how this information about a schema can be used to make conclusions about the design. Some illustrations are also provided. We conclude with an analysis of the strengths and weaknesses of this approach and the plans for future work.

3. THE PARADIGM

In the spirit of Steel (1986), we first define the language of discourse of this paper. It is assumed that databases are being "semantically" defined. That is, there are object types, objects (instances of object types), relationship types, relationships (instances of relationships) between objects, and constraints. Relationships can be realized through the use of a variety of constructs (Peckham and Maryanski 1988). In the tradition of semantic database modeling, these types, relationships, and constraints are specifiable at database design time.

The most general definition of the word "constraint" is taken since many types of constraints can cause compensating actions which are then propagated throughout the database. Associated with each constraint is a *propagation rule*. The propagation rule specifies the compensating actions that must be taken on the database in order to maintain the constraint, and in a larger sense database consistency and correctness. The semantics of a database can then be interpreted as the constraints and the compensating actions needed to maintain them.

For example, suppose that in a management information system we have PRODUCTION objects representing products to be manufactured by a firm and PRODUCTION_STATS objects recording pertinent manufacturing statistics about associated products. The following is an example of a constraint.

A PRODUCTION_STATS object may not exist without an associated PRODUCTION object.

One example of a propagation rule which might be defined to maintain the constraint follows.

Upon deletion of a PRODUCTION object, the associated PRODUCTION_STATS object must also be deleted.

Leveson, Wasserman, and Berry (1983) have pointed out that it is important for the propagation rules to be consistent with the constraints with which they are associated. If propagation rules are specified at design time, then a check can be done to insure these are consistent with the constraints of the system. This is the approach taken with their system, BASIS. However, we note that this process can be time consuming and complex.

For the purposes of this investigation, we categorize constraints and for each constraint type give a list of possible propagation rules which may be enforced to maintain the constraint. We assume that the designer chooses one of these compensating actions for each constraint when designing the database.

For example, consider the constraint presented above. It is clear that it is possible to violate this constraint in the following ways.

(1) The insertion of a PRODUCTION_STATS object.

(2) The deletion of a PRODUCTION object.

(3) The modification of one of the objects in a manner that impacts the relationship.

(4) The manipulation of an object establishing the relationship between the PRODUCTION and PRODUCTION_STATS objects.

Should the insertion of an PRODUCTION_STATS object violate this constraint, the denial of the operation or the simultaneous insertion of the associated PRODUCTION object are compensating actions which will insure the maintenance of the constraint. Similarly, we can establish the only possible compensating actions or propagation rules for the remaining three operations. Later, we show that once each constraint type is carefully specified we are able to carry out a similar analysis to enumerate possible propagation rules.

Thus, for the purposes of this investigation, we define "propagated behavior" in a database as the *compensating actions which are specified, during database design for the purpose of maintaining the constraints of the database schema*. It is the purpose of this research to define a methodology to capture constraints and the resulting propagated behavior of a given semantic model which can then be used as an aid in describing databases which will remain consistent and valid once they are instantiated.

What we describe is a means by which the propagation semantics of a database could be captured for analysis by the designer. Constructs were chosen to provide the representation of a broad range of database semantics. Only those constructs necessary for the specification of the information needed to perform the desired analysis will be presented. This is different from a model used to specify a complete semantic database.

The constructs needed for this analysis are *primitive types*, *attributes*, *primitive operations*, *constraints*, and *propagation rules*. In order to achieve our goal of model independence, we refrain from using more complex constructs. In order to utilize these ideas for a particular model, we envision the decomposition of the constructs of the model to these primitive constructs. One of the goals of future research is to examine predominate constructs from well known semantic and object oriented models and to illustrate their representation using our paradigm.

PRIMITIVE TYPE: Primitive object types are the types upon which range constraints may be defined. The ranges of the primitive types are subsets of the primitive types of the system upon which the model is defined. In a traditional system, the primitive types might be string, integer, and boolean. In other systems they might be file, syntax-tree, etc.

OBJECT TYPE: An object type, OT, is a 2-tuple

OT = <A, R>

where

A is an aggregation of attributes

R is a set of rules, each of the form <C,P>,

where C is a constraint

and P is a propagation rule.

ATTRIBUTE: An attribute is a descriptor of a property of an object. It may take the form of either a primitive type or a set of types. An attribute, A, of an object O, is denoted O.A. Key attributes of an object uniquely identify the object.

For the remainder of the paper, when the word attribute is used it is assumed to mean "non-key attribute" unless otherwise specified. Without loss of generality, we assume database systems will not support modification of key attributes.

PRIMITIVE OPERATION: The following are the primitive operations of interest: insert an object, delete an object, modify a non-key attribute, and read an attribute. Unless a constraint specifies otherwise, it is assumed that these operations may be applied to all instances of the constructs for which they are defined.

In order to manipulate a database, higher order operations must be defined. For example, procedures may be defined to manipulate data, methods may be defined as operations associated with particular database object types, or transactions may be defined as atomic units of manipulation over the database. All higher order operations must be constructed from primitive operations, which represent the only means by which database instances may be manipulated. Here we view these primitive operations as the initiators of the propagated actions of interest in this research.

CONSTRAINT: Constraints specify the required conditions of the database.

PROPAGATION RULE: A propagation rule specifies the compensating actions the database must perform in order to maintain the conditions specified by a constraint.

In the following sections, we classify constraints as range constraints, derivation constraints, and existence constraints. Since propagation rules have a natural association with constraints, for each constraint type we examine the propagation rules which might be associated with it. The propagation rules for a given constraint are generated by observing how primitive operations might violate the constraints when applied to objects addressed by the constraint. Possible compensating actions are then enumerated. It is assumed that all database operations, procedures, methods and/or transactions use only these primitive operations to manipulate the database. In a system containing primitive operations or constraint types which differ from the primitive operations given here, a similar analysis would be required to determine the associated classes of propagation rules for each constraint type.

3.1 Range Constraints

A **range constraint** is the definition of the allowable range of an attribute. A range constraint may be the definition of the range as a primitive type, a set of primitive types, an object type, or a set of object types. This may include the specification of the size of the set. Attribute constraints are also range constraints.

An attribute constraint (Dogac, Chen, and Erol 1985) is the specification of the range of an attribute based upon states of other attributes and/or objects in the database. An example is the specification in a management information system that the Expected_weekly_production attribute divided by the Av_daily_production attribute may not be greater than five for a given ASSEMBLY_LINE object, i.e., an assembly line may not be expected to put out more of a given product than past experience indicates could be produced.

Specification that an attribute is of object type OT means that an association is established between the object carrying the attribute and one instance of an object of type OT in the database. This has characteristics somewhat like an existence constraint, since the existence of the associated reference object is required. However, because it also strongly resembles the other range constraints in that it determines permissible attribute values, we have chosen to classify this as a range constraint.

Recall that compensating actions associated with a given constraint are not unique. Thus, for each constraint type we must also list the possible propagation rules. Consider an object O of type OT containing an attribute A. The following operations must be considered.

(1) Consider the insertion of an object O of type OT containing the attribute A or the modification of A. If the insertion of O or the modification of A violates the range constraint, then one of the following propagation rules may be given.

   a) Deny the insertion (modification).

   b) Automatically insert another related object so that the insertion or modification of this object does not violate the constraint. (This refers to the case in which the range of the attribute has been defined as an object type and thus the attribute references other objects in the database. Further explanation will be given below.)

c) Insert O, but automatically change attribute A to "null."

(2) Upon deletion of an object O1 of type OT1 which is associated with attribute A of object O, the following propagation rules may be applied. (In this case, assume the range of A is declared to be of type OT1, or think of A as referencing objects of type OT1.)

a) Deny deletion.

b) Delete and also delete O.

c) Delete and change O.A to "null."

Inserting an object of type O (or modifying the attribute, A) and marking the value "dirty" will not be considered as violating a range constraint. Instead, in terms of the response of the system, it is considered as an attribute with a wider range (clean and dirty values) and another attribute derived from it (clean or dirty).

The issuance of a denial to the user is considered a termination of possible propagation. Thus it is not included as a meaningful result. The user's later attempt to insert the object or modify the attribute is considered the initiation of a new action, not propagated by the old one.

An example may help to explain rule 1b above. Using our example of PRODUCTION_STATS and PRODUCTION objects, assume that a constraint specifies that the Product attribute of a PRODUCTION_STATS object may not reference a non-existent PRODUCTION object. The associated propagation rule may specify that upon insertion of a PRODUCTION_STATS object, an associated "empty" PRODUCTION object must also be inserted. Since the attributes of the PRODUCTION object may not have been specified yet, they must be set to "null." Since the PRODUCTION_STATS object is referencing a PRODUCTION object, enough information is present to insert the otherwise null object.

3.2 Derivation Constraints

A **derivation constraint** is the specification of a function F:OT1: x ... x OTn → OT.A, where OT1,...,OTn are object types, and OT.A is an attribute of object type OT.

Thus, a derivation constraint is the specification of the means by which one attribute in the database is computed from other attributes in the database. An example is the specification that PRODUCT.Total_cost is the sum of cost attributes from associated PLANNING, ENGINEERING, MANUFACTURING, SALES, and SERVICE objects.

Let us assume a derivation constraint in which an attribute A in an object O is being functionally derived from the attributes A1,...,An, in the objects O1,...,On of types OT1,...,OTn, respectively, by the function F:A1 x...x An → A. The propagation rules which may be associated with derivation constraints are enumerated as follows.

(1) Upon insertion of object O, compute the derivation.

(2) Deny modification of attribute A if it violates the derivation rule.

(3) In the case of deletion of one of the objects O1,...,On, related by the derivation constraint to an object O.

a) Deny the deletion.

b) Compute the derivation function. (With this choice of propagation rule, it is assumed that the derivation function has been defined in such a way that it can gracefully handle incomplete information.)

(4) Upon modification of one of the attributes A1,...,An or the insertion of one of the objects containing these attributes, the following propagation rule must be given.

a) Modify O.A by the derivation function.

In the second above we are stating that the user of a system should not be permitted to modify a derived attribute A. Since A is derived from other attribute values in the system, its values cannot be user defined.

3.3 Existence Constraints

An **existence constraint** is a rule which specifies under what conditions an instance of an object may or may not exist. An example of an existence constraint follows. An object O1 of type OT1 may not exist without the existence of an object of type OT2 with attribute A referencing O1.

Existence constraints represent relationships between object types established through attributes. The following operations are pertinent to existence constraints.

(1) In the case of the insertion of an object of type OT1, one of the following propagation rules may be chosen.

a) Deny the insertion unless the related object is present.

b) Automatically insert the related object of OT2 with attribute A referencing O2.

(2) In the case of the deletion of an object of type OT2, or the modification of attribute A, the following rules are possible choices.

a) Automatic deletion of any related objects of type OT1.

b) Denial of the deletion or modification.

Again, rule 1b assumes sufficient information is available to make the appropriate automatic insertions.

In this paper, a complete set of formal proofs of the consistency of the constraints with their associated propagation rules is not given. Since the proofs will all be of the same general type, one example is given to illustrate the proof technique which will be used in later work. For our example, we choose the following derivation constraint and one of its associated propagation rules.

Derivation constraint: F:OT1 x ... x OTn → OT.A, where OT1,...,OTn are object types, and OT.A is an attribute of object type OT.

Propagation Rule: In the case of deletion of one of objects O1,...,On, related by the derivation constraint to an object O.

(1) Deny the deletion.

(2) Compute the derivation function. (With this choice of propagation rule, it is assumed that the derivation function has been defined in such a way that it can gracefully handle incomplete information.)

Theorem: The derivation constraint and the corresponding deletion propagation rules given above are consistent. That is, the propagation rules will correctly maintain the consistency of objects related thorough the derivation function under the deletion of objects in the domain of the derivation function.

Proof of consistency: We wish to show that the propagation rule will uphold the constraint under the deletion of one of the objects Oj, where j is 1,2,..., or n. We assume that the database maintains the constraint previous to the proposed deletion. We wish to show that each of the two actions will maintain the constraint. In the case of denial of the deletion, the state of the database will not be changed, and thus the database remains consistent. In the case of derivation of O.A using the function F, the derivation constraint will clearly be maintained.

With this section, we complete the description of the paradigm used for representation of the modeling concepts pertinent to propagation of compensating actions in a database. Notice that the concept of "relationship" is absent in an explicit sense, since the same concept can be presented through the definition of constraints and propagation rules. Although this uniform definition of database semantics is convenient for propagation analysis, the inclusion of relationships, IS-A hierarchies, etc., may be a more appropriate vehicle for capturing the semantics of the database from the user.

In the next section, we examine the representation of constraints and propagation rules used to make helpful conclusions about the information specified about the database.

4. USING THE PARADIGM

The paradigm given in Section 3 shows how the structure, constraints, and propagation rules of a particular database can be captured. The next step is to illustrate how this information can be used to reason about the database. We now turn our attention to some examples from a hypothetical management information system. A first-order logic representation for the constraints and propagation rules will be used for convenience of representation and computation.

Note that, as expressed below, propagation rules do not specify the details of the manipulations of operations upon objects. This is a detail that would be captured by the model used for design. For example, suppose a PRODUCT.Cost_to_date attribute is functionally derived from other information in the database. The right-hand side of an associated propagation rule might read Modify (Cost_to_date). The details of this Modify might be captured in a method, Compute_new_cost, at the design-model level. However, the only pertinent information for this design-time analysis of propagated actions is that the Cost_to_date attribute is being modified.

Let O, O1,...,On, be objects of types OT, OT1,...,OTn, respectively, and let A, A1,...,An be attributes of the respective objects. Let OP represent a primitive operation and OP(O) represent the application of operation OP on object O. Also let Oi.A R Oj represent an association, R, between an object Oi of type OTi and an object, Oj, of type OTj, through attribute A in Oi.

In order to logically represent propagation types, we notice that they are generally of two types. One represents the case in which we are denying a proposed action, OP, unless constraint C is maintained. This can be represented as follows.

$$\text{NOT } C \rightarrow \text{NOT } OP(O)$$

The second type of rule is used to capture the situation where an action on one object propagates another action on the same object or another object or attribute.

$$OP1(O1) \rightarrow OP2(O2),$$
where O1.A R O2, or O2.A R O1.

Among the conclusions which we are able to draw from the extracted modeling constructs, there are two types in which we are interested.

(1) Direct conflicts in definition.

(2) Implicit and/or redundant semantics.

Both arise from situations in which the modeler has defined relationships between object types, focusing on one at time, and may need help to keep track of the implications of these definitions and their interrelationships. A partial enumeration of types of conflicts and explicit and/or redundant semantics is give in the following sections.

4.1 Direct Conflicts

Direct conflicts represent situations in which the semantics of one object or attribute are in conflict with the semantics of another object or attribute. Two examples follow.

(1) Conflicting attribute definitions: The specification of an attribute as both derived and having a range constraint may produce a conflict. It is especially clear that if the range of the derivation function does not intersect the domain definition of the attribute, there is a conflict.

(2) Conflicting propagation rules involving more than one object: For example, suppose the designer specifies a propagation rule following from a constraint on object type OT1 which impacts on object type OT2. Also suppose that from a constraint on OT2 a propagation rule impacting OT1 is defined. Then, as a result of these definitions, a conflict may have been produced.

For an example of the first type of direct conflict, we consider PERSON, ENGINEER, and MANAGER object types in the engineering department of a corporation. Let us assume that existence constraints specify that for every instance of a MANAGER, there must also be an associated instance of type ENGINEER, and that for every instance of type ENGINEER, there must be an associated instance of type PERSON. Also assume that ENGINEER.Salary is derived (inherited) by way of the identity function from PERSON.Salary. Similarly, MANAGER.Salary is derived (inherited) from ENGINEER.Salary. (These constraints represent the semantics which might be included in the specification of an IS-A hierarchy of types, in which ENGINEER IS-A PERSON and MANAGER IS-A ENGINEER.) Let us further assume that while modeling the Manager type, the designer has specified the following attribute constraint. MANAGER.Salary must be greater than ENGINEER.Salary for all instances of ENGINEER. This will readily produce a direct conflict, since the derivation function specifies that MANAGER.Salary is equal to ENGINEER.Salary.

An example of the second type of direct conflict is illustrated as follows. Suppose an object of type OT1 cannot be inserted without an associated incidence of an object of type OT2 and vice versa. For example, suppose that upon insertion of a PERSON object into a database, a PERSONNEL object for maintaining personnel records must also be inserted. Similarly, upon insertion of a PERSONNEL object, the associated PERSON object must also be present. The designer should be reminded of this situation. The designer will then be able to determine if this will be handled by using, say, transactions to assure objects of both types are (logically) inserted simultaneously or if one propagation rule should be changed to prevent the conflict. This type of conflict could arise if constraints are specified for each object at different times in the design process (or perhaps by different designers on a large project).

4.2 Propagated and/or Redundant Semantics

This classification represents situations in which there are no direct conflicts in semantics, but in which awkward or redundant definitions are evident or in which propagated semantics that should be called to the attention of the designer are present.

(1) Transitive closure: This represents propagated semantics derived from the transitive-closure of the designer specified semantics. For the purpose of this analysis it is convenient to view any primitive operation as a manipulation, independent of whether it is an insertion, deletion, or modification. We can then compute all possible sequences of actions which might be propagated by individual actions on the database. Simply stated, we have the following transitive property.

Given object types OT1, OT2, and OT3, and primitive operations, OP1, OP2, and OP3, the following holds.

If OP1(O1) → O2(O2) and OP2(O2) → OP3(O3) are propagation rules, then OP1(O1) → OP3(O3).

(2) More than one propagation rule is defined on a pair of objects: Although there is no direct conflict in definition, it is possible the designer has specified these through different views on the objects and might wish to consolidate and/or clarify the relationship between the two objects.

An application of the transitive-closure property follows. Suppose there are EMPLOYEE, ACTIVITY, and MANAGER objects in a management information system. ACTIVITY objects record the professional activities of a given EMPLOYEE within the organization and MANAGER objects record additional information about employees which are also managers. The attribute EMPLOYEE.Stipend is derived as a function of the activities in which the employee is involved and records the additional monthly stipend the employee receives above and beyond the monthly paycheck. The Stipend attribute of a

Manager object is derived by way of an identity function from the related EMPLOYEE object.

We now suppose that the usual constraints on derived attributes hold and the following propagation rules have been specified.

Modify (ACTIVITY.Time_consulting) →
Modify (EMPLOYEE.Stipend)

Modify (EMPLOYEE.Stipend) →
Modify (MANAGER.Stipend)

If we apply the transitive property to these propagated actions, we get the following.

Modify (ACTIVITY.Time_consulting) →
Modify(MANAGER.Stipend)

The designer is then notified that modification of the Time_consulting attribute of ACTIVITY objects will result in the modification of the Stipend attribute of MANAGER objects. But now suppose that although managers can derive additional income from other activities, the organization does not permit managers to derive additional income from consulting activities. Presenting this propagated activity in the database to the designer may serve as a reminder that the derivation of MANAGER stipends must be different from that of other employees.

There are some instances in which a database design system may not wish to present all transitively derived propagated sequences to the designer. For example, within an IS-A hierarchy, the transitivity of inheritance is properly conveyed by the graph structures usually presented to the designer. That is, the designer will not want to be reminded that whenever A IS-A B and B IS-A C, then A IS-A C. However, when attribute values are derived from other information in the schema through different relationship structures, which were defined during different phases of the specification process, this derived information may provide meaningful feedback.

The handling of cycles is a problem which is not fully investigated here. However, we do consider cycles of length two in which two objects may be connected through propagation rules. For example, suppose there are PERSON and INSURANCE objects representing employees and information about their families. Further assume that PERSON and INSURANCE objects reference each other through their attributes and that the following propagation rules have been given by the designer when constructing each of the object types.

(1) Employee attribute of an INSURANCE object must reference an existing PERSON object.

(2) Insurance attribute of a PERSON object must reference an existing INSURANCE object.

Thus, propagation rules associated with these constraints specify that INSURANCE and PERSON objects must be simultaneously deleted. It may be the case that the designer wishes to consolidate these two rules into one rule handling the deletion of both types of object. One benefit might be the simplification of application code. Here, for example, there needs to be only one procedure written to handle the deletion of both types of objects.

5. THE CONCLUSION

Recent research has proposed several modeling constructs and methodologies which can be used for the extraction of information from the database modeler. While these techniques provide several tools for the representation of "real world" objects and relationships, the expression of the actions which necessarily follow from these semantics has often been left out. Thus, we have investigated the means by which more semantic knowledge might be incorporated into a database schema. This paper represents the first phase of our attempt to provide this design-time analysis of databases.

As for our future work, we plan to undertake a further analysis of constraint types. For example, a study of the specification and representation of cardinality constraints was not sufficiently explored here. Transitional constraints (Dogac, Chen, and Erol 1985), whereby the next state of an object is dependent upon the current state of that and other objects, should also be explored. Finally, we must demonstrate that this paradigm of representation and computation is rich enough to be used with a wide range of existing semantic models.

A second area of interest is the detection of semantics encoded in the methods and transactions of a database system. Since actions, transactions, or operations may be defined at design time, conditional behavior coded in these operations may also be construed as defining constraints and associated propagation rules. An example follows.

Operationally Defined Constraint and Rule

```
If Average(ENGINEER.Salary) > Average(MANAGER.Salary
    then for every X = Manager,
        Modify(X, Equalize_salary, Average_engineer,
            Average_manager_salary).
```

The code in the example specifies that, on the average, engineers' salaries may not exceed managers' salaries. This is an attribute constraint on the ENGINEER object type. Also included here is the propagation rule which dictates the action to be taken upon the violation of the constraint. It might be possible to perform an evaluation of transactions written at run time against the constraints and propagation rules written at design time to determine, before the transaction is used, if there will be conflicts.

Work related to this has been carried out by (Ngu 1989), in which static modeling information is inferred from transactions as they are being designed.

Constraints reflecting rules for instantiation of objects and associations between them is also needed to completely describe the semantics of relationships such as IS-A and classification. This characterization will be necessary for two reasons. First, a more meaningful analysis of possible cyclic relationships between types could be conducted if some information about the instance level of the database is known. Since this research is concerned with design time analyses, it is possible to examine only the rules dictating the materialization of instances. Second, a better analysis of propagated semantics can be carried out if we do not have to examine the worst case scenario for each constraint. Thus, the rules for propagation could take the form

$$OP1(O1) \times IR \rightarrow OP2(O2)$$

where OP1 and OP2 are primitive operations, O1 and O2 are objects, and IR is a rule dictating the instantiation of objects and/or associations between them. In this way, we can examine under what conditions the propagated action will or will not take place.

Future work will also involve the formalization of the representation and analysis of database constraints and propagation rules. This effort will focus upon the identification of a formal structure capable of faithfully representing database enterprises and capable of simplifying the analysis of the database schema.

The approach outlined in this paper for the detection of database inconsistencies is meant to complement current developments in semantic and object-oriented database design. A representation of database semantics through the use of specific types of constraints and their associated propagation rules has been developed independent of individual modeling constructs and methodologies. This effort illustrates an attempt to represent and perform computations on the constructs and rules of a database system and to provide results which would generate useful feedback to the database designer.

6. ACKNOWLEDGEMENTS

The authors would like to thank Sanjay Ramamurthy for the helpful proofreading and comments during the development of this paper.

The work of Joan Peckham was partially supported by an IBM teaching fellowship.

The work of Fred Maryanski was partially supported by grant IRI-8704042 from the National Science Foundation.

The work of Steven A. Demurjian was partially supported by grant 1171-000-22-0506-35-038 from the University of Connecticut Research Foundation.

7. REFERENCES

Abiteboul, S., and Hull, R. "IFO: A Formal Semantic Database Model." *ACM Transactions on Database Systems*, Volume 12, Number 4, December 1987, pp. 525-565.

Brodie, M., and Ridjanovic, D. "On the Design and Specification of Database Transactions." In M. Brodie, J. Mylopoulos, and J. Schmidt, Editors, *On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases, and Programming Languages*, New York: Springer-Verlag, 1984, pp. 277-332.

Carlson, C., and Adarsh, K. "Toward the Next Generation of Data Modeling Tools." *IEEE Transactions on Software Engineering*, Volume 11, Number 9, September 1985, pp. 966-970.

Dittrich, K. "Object-Oriented Database Systems: The Notion and the Issues." In K. Dittrich and U. Dayal, Editors, *Proceedings, 1986 International Workshop on Object-Oriented Database Systems*, pp. 2-4.

Dogac, A.; Chen, P.; and Erol, N. "The Design and Implementation of an Integrity Subsystem for the Relational DBMS RAP." *The Fourth International Conference on Entity Relationship Approach*, October 28-30, 1985, Chicago, Illinois, pp. 295-302.

Hull, R., and King, R. "Semantic Database Modeling: Survey, Applications, and Research Issues." *ACM Computing Surveys*, Volume 19, Number 3, September, pp. 201-260.

King, R., and McLeod, D. "A Unified Model and Methodology for Conceptual Database Design." In M. Brodie, J. Mylopoulos, and J. Schmidt, Editors, *On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases and Programming Languages*, New York: Springer-Verlag, 1984, pp. 313-327.

Leveson, N.; Wasserman, A.; and Berry, D. "BASIS: A Behavioral Approach to the Specification of Information Systems." *Information Systems*, Volume 1, Number 3, 1983, Pergamon Press.

Mylopoulos, J.; Bernstein, P.; and Wong, H. "A Language Facility for Designing Database-Intensive Applications." *ACM Transactions on Database Systems*, Volume 5, Number 2, June 1980, pp. 185-207.

Ngu, A. "Transaction Modelling." *Proceedings, Fifth International Conference on Data Engineering*, Los Angeles, California, February 1989, pp. 234-241.

Peckham, J., and Maryanski, F. "Semantic Data Models." *ACM Computing Surveys*, Volume 20, Number 3, September 1988, pp. 153-189.

Steel, T. *A Minimal Conceptual Schema Language for Life, the Universe, and Everything, Database Semantics* (DS-1), T. Steel and R. Meersman, Editors, IFIP, 1986, pp. 255-284.

Urban, S. *Constraint Analysis for the Design of Semantic Database Update Operations*. Unpublished Ph.D. Dissertation, The Center for Advanced Computer Studies, University of Southwestern Louisiana, September 1987.

Urban, S., and Delcambre, L. "Constraint Analysis for Specifying Perspectives of Class Objects." *Proceedings, Fifth International Conference on Data Engineering*, Los Angeles, California, February 1989, pp. 10-17.

Wilson, G. "A Conceptual Model for Semantic Integrity Checking." *Proceedings of the Sixth International Conference on Very Large Data Bases*, 1980, pp. 111-125.