AN EFFICIENT OODB MODEL FOR ENSURING THE INTEGRITY OF USER-DEFINED CONSTRAINTS

Belal Zaqaibeh[1], Hamidah Ibrahim[2], Ali Mamat[2], and Md Nasir Sulaiman[2]
[1]*Faculty of Information Technology, Multimedia University, 63100 Malaysia,*
[2]*Faculty of Computer Science and Information Technology, University Putra Malaysia*

Abstract: In this paper, a new structural model is proposed to ease the checking of the integrity constraints in an object-oriented database system. The structure accepts declarative global specification of constraints including user-defined constraint, and an efficient representation that permits localized constraints checking. A new method called "Detection" is added to the structure to check the status of violation of the relations in an object-oriented database. The new approach is demonstrated using ALICE rule.

The notion of the constraints is used to define the connectivity between objects required for the valid expression of constraint and rule conditions. The event in Integration Rules (IRules) that defines the active behavior of an application specifies an operation to be monitored, such as modifying a data value. The semantic analysis process applies a concept known as object-centered conditions during the compilation of ALICE rules to detect semantically incorrect rules at compile time.

Keywords: Constraint, Maintenance, Object-Oriented Database, ALICE, Inter-Object Constraint.

1. INTRODUCTION

Object-oriented databases are rapidly gaining popularity, and show a promise of supplanting relational databases. It is imperative that explores the maintenance of integrity in object-oriented databases. By virtue of object

orientation, some integrity constraints are represented naturally and maintained *for free* in an object-oriented database, the system type and the object class hierarchy will directly captured. Typical example of this sort is the constraint that every employee is a person and that every child of a person is a person. Other forms of integrity constraints apply to a single object, and clearly belong as part of an object class specification. An example of such a constraint for a person object is that years-of-schooling must be at least 5 years old.

As known, the integrity constraints that involve the monitoring of user updates on data items in a single object called as intra-object constraint. The following example shows the domain constraints that specify legal values in a particular domain: *Sex in* [*F, M*]. In other hand the integrity constraints that involve objects from more that one class known as inter-object constraints. For example of inter-object constraints is the association between *student* and *course* classes is expressed as each student must register in at least three courses, it will be represented as *count (student.course) >= 3,* assuming that *course* in *student* is a data structure that holds the object identifiers of the courses taken by the particular student.

Traditionally, integrity constraints in object-oriented database systems are maintained by rolling back any transaction that produces an inconsistent state for the intra-object constraint, or disallowing or modifying operations that may produce an inconsistent state for the inter-object constraint. An alternative approach is to provide automatic repair of inconsistent states using production rules. For each constraint, a production rule is used to detect constraint violation and to initiate database operations that restore consistency. The maintenance system consists of a set of constraint services with different solving capabilities and complexity. Each service can be connected to maintain constraint relationships independently.

Object-Oriented Database Systems (OODBs) have been designed to support large complex programming projects. OODBs divide the definition and maintenance of all structures into inheritance encapsulated method, with information shared between classes kept to a minimum, since maintenance required code in multiple classes. The monitor specific state is changed in instances of the external classes. This is in contrast with relational database systems, which were designed to provide a large data repository accessible to the user through a general purpose, and declarative-style query language. Declarative query languages are well suited to handling arbitrary queries presented by an end user, but they introduce a burdensome impedance mismatch when embedded within application code. A language for the expression of Integration Rules (IRules) is an important part of an Object-Oriented Database Management System (OODBMS) that defines the active

behavior of an application [7,11]. The event in an IRules specifies an operation or a situation to be monitored, such as modifying a data value.

2. PRELIMINARIES

The collection objects are the elements that allow an object to contain multiple values of a single property. The collection objects identified by the proposed standard including *set* that contains an unordered group of objects of the same type, no duplicates are allowed so this will help to reduce the number of constraints checking when an event happens, *bag* contains an unordered group of objects of the same type, duplicates are allowed, *list* is an ordered group of objects of the same type, *array* is an ordered group of objects of the same type that can be accessed by position, and *dictionary* is like an index [1, 3, 7]. Collection objects are made up of ordered keys, each of them is paired with a single value. From the collection objects a search can be performed using the keys to find the full information about any object in the database. Those keys are known as Object Identifiers (OID). OID is an internal database identifier for each individual object and this might include the page number and the offset from the beginning of the page for the file in which the object is stored [2, 3, 7].

The major types of classes used in object-oriented are control classes which manage data and have visible output, it controls the operational flow of the program [1, 2, 7]. Entity classes are used to create objects that manage data. Most object-oriented programs have at least one entity class from which many objects are created. In fact, in its simplest sense, the object-oriented data model is built from the representation of relationships between objects created from entity objects [2, 11]. Container classes existed to "contain" or manage, multiple objects that created from the same type of class [2]. Because they gather objects together, they are also known as aggregations.

Entity integrity normally enforced through the use of a primary key or unique index [2, 3]. However, it may at times be possible to have a unique primary key for a tuple (row) a relation and still have duplicate data in its' fields. This simply means that in any given relation, every tuple is unique. In a properly normalized, relational database, it is of particular importance to avoid duplicate tuples in a relation because users will expect that when a tuple is updated, there are no other tuples that contain the same data.

In the domain integrity the values in any given column fall within an accepted range [2, 3]. It is important to make sure that the data entered into a relation is not only correct, but also appropriate for the columns it is entered into. The validity of a domain may be as broad as specifying only a

data type (text, numeric, etc.) or as narrow as specifying just a few available values. Referential integrity, here is the foreign key value points to valid rows in the referenced table or points to, a related tuple in another relation. It is absolutely imperative that referential integrity constraints be enforced [6, 8]. While it is possible that foreign key values may be null, it should never be invalid. If a value of foreign key is entered, it must reference a valid row in the related relation.

There are two models that deal with object-oriented database; Abstract Data Type (ADT) and Object-oriented System Model (OSM). The proposed approach is built on OSM because it has some similar features to formal syntax and semantics based on a temporal, first-order logic and it allows for multitude of expressive, and high-level view [2].

3. OUR CONSTRAINTS DOMAIN

In general there are two types of integrity constraints: static constraint and dynamic constraint. The static constraints are known and controlled by the OODBMS in addition to some dynamic constraints. Most of the dynamic constraints are complicated and unpredictable because it depends on the users needs. Figure 1 shows that the subclass *employee* inherits *classes* and *details* from *manager* and *secretary* in which both inherited from their superclass *name.*

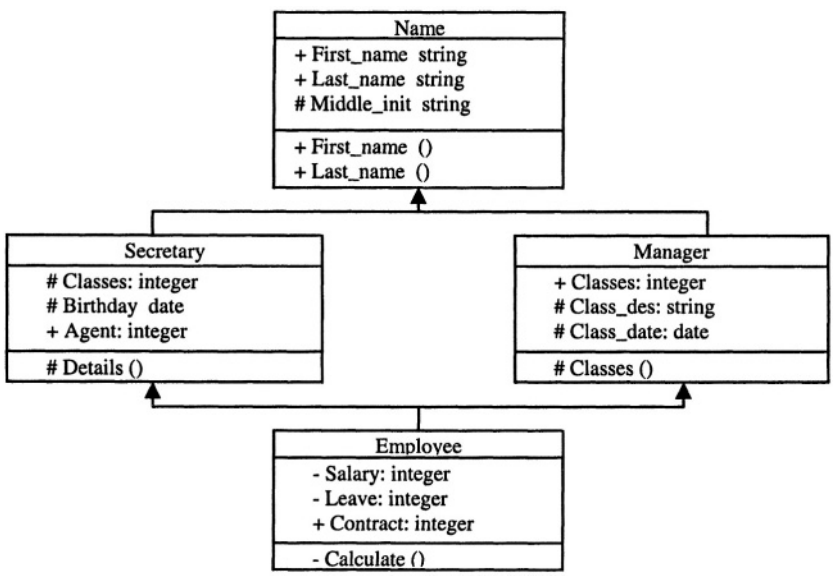

Figure1. The multi inheritance relationship

4. RELATED WORK

The most relevant researches on expression capabilities for active rule languages that are designed to maintain the database integrity. High Performance ACtive DBMS (HiPAC) is one of the first projects to extensively address active database issues and it is proposed for coupling [7, 9]. Active DBMS allows rules to be fired and executed automatically. One of the first occurrences of active capabilities was the use of ON conditions. Triggers and assertions are used to maintaining integrity constraints [5].

ARIEL rules are based on the relational data model extended with a production rule system. Rules in ARIEL are triggered based on specific events or pattern matching, as in expert systems. Rule condition testing in ARIEL is implemented by use of a variation of the Rete algorithm to improve performance [5, 7, 9].

The Object Database and Environment (ODE) represents some of the more efforts in the development of active rule languages [5]. Unlike most of the other systems, it supports a rule language within an OODB, the expression of constraints which are triggered by database updates, and the expression of rules triggered by condition monitoring [9].

STARBURST also represents one of the more developments in active rule languages [9]. Conditions in STARBURST are expressed by use of SQL with additional syntax for the expression of events, conditions, actions, and rule priorities.

Based on the active rule languages described above, a summary of the basic features associated is presented in the Table 1.

Table 1. Comparison of constraint rules

Features	HiPAC	ARIEL	ODE	STARBURST
**** Temporal events***				
Absolute points	✓	✓	✓	✓
Relative points	✓		✓	✓
Periodic points	✓	✓	✓	✓
**** Integrity of constraints***				
State constraints	✓	✓	✓	✓
Transition constraints	✓	✓		
**** Conditions***				
Quantified variables				✓
Negated conditions	✓	✓	✓	✓
Aggregate functions	✓			✓
User-defined functions	✓	✓	✓	✓

Features	HiPAC	ARIEL	ODE	STARBURST
**** Actions***				
Single database operations	✓	✓	✓	✓
Compound operations	✓	✓	✓	✓
User-defined operations	✓	✓	✓	
**** Triggers***				
Event	✓	✓	✓	✓
Pattern		✓	✓	
**** Data model***				
Relational		✓		
Extended relational				✓
Object oriented	✓		✓	

Assertion Language for Integrity Constraint Expression (ALICE) is an expression active rule language that is designed to maintain constraints over the expression of complex database.

5. OVERVIEW OF ALICE

ALICE was developed as a declarative constraint language for the expression of complex and for stating constraints in an object-oriented environment, and logic-based constraints in an object-oriented environment. As in an object model, it assumes that the existence of objects with unique object identifiers. Objects of a similar type are organized into classes, which are organized into ISA relationships; one class is a subclass of another class. If an object is an instance of a subclass, then the object must also be an instance of all of its superclasses [5,7]. The immediate subclasses of a superclass can be specified as disjoint subclasses as a way of imposing additional constraints on the classes in which an object participates.

Each class is described through the use of property definitions. Properties can be single or multi-valued. Inverse property relationships are also supported. As additional semantic detail, properties can be specified as required (no null values) and/or unique (establishing a one-to-one relationship between an object and its property values). A subclass inherits all of the property definitions of its superclasses [4, 5, 7].

ALICE provides a tool for expressing generalized, logic-based constraints against an object-oriented schema. In addition, its constraints are analyzed by a constraint explanation tool (CONTEXT) and are subsequently transformed into active database rules. The rules generated by CONTEXT provide a way to recover from constraint violations at execution time and also provide tools for translating constraints into active database rules [7, 9].

An important aspect of ALICE is representing a rule language that can be applied within an object-oriented model of data. Figure 2 shows the syntax of ALICE rule.

rule-name [IN *rule-set-name*] [LEVEL *n*] IS
EVENT: *database-operation* | user-defined-operation | *time-specifier*
[CONDITION: *logical-expression*| *quantification*]
ACTION: {*sequence-of-operations*}
END OF RULE

Figure 2. **ALICE rule format**

ALICE is strictly a constraint language [7, 9]. It cannot be used to directly express constraints in rule form, and does not support the expression of rules in general. It is restricted to the expression of static constraints among objects. The condition expression capabilities of ALICE are also limited because the original work on ALICE focused on the mapping of ALICE constraints to first-order logic [7, 9]. Also it does not support the expression of transition constraints or the use of external functions for complex conditions.

6. METHODOLOGY

The proposed structure exhibits some properties as it provides the capability to represent complicated relationships as it is based on a generic object system, so that some special relationships can be specified and maintained uniquely such as the relationships between domains and tasks. It can also maintain relationships between a set of distributed objects. This object model makes it possible to implement concurrent or distributed maintenance services. When the sets of objects or the constraints relationships of different types are completely independent, the constraints can be maintained by multiple processes simultaneously. This improves performance, which makes the new model constraint system easy to extend and can be integrated with any existing or specialized constraint services.

For example, assume that some information about two types of employees in a company is needed. The first type is a manager with (ID, name, classification ID, classification description, and classification date). The second type is a secretary with (ID, name, classification ID, birthday, and recruiting agent). Figure 3 shows the data definition language of the above example.

```
CREATE TYPE Name AS OBJECT (
 First_name char (15),
 Last_name char (15),
 Middle_init char (1);
 MEMBER PROCEDURE initialize);

CREATE TYPE BODY Name AS
    MEMBER PROCEDURE initialize IS
 BEGIN
          First_name := NULL;
          Last_name := NULL;
 END;

CREATE TYPE Secretary AS OBJECT (
 Classes integer,
 Birthday date,
 Agent integer,
 S_Name Name;
 MEMBER PROCEDURE initialize);

CREATE TYPE BODY Secretary AS
   MEMBER PROCEDURE initialize IS
 BEGIN
          Details := NULL;
 END;

CREATE TYPE Manager AS OBJECT (
 Classes integer,
 Class_description char (15),
 Class_date date,
 M_Name Name;
 MEMBER PROCEDURE initialize);

CREATE TABLE Employee (
 Person_ID integer,
 Recruiting_agent integer,
 Classes Manager,
 Details  Secretary,
 PRIMARY KEY (Person_ID),
 FOREIGN KEY (Classes) REFERENCES Manager);
```

Figure 3. A relation and its constraints

As shown earlier in Figure 3, the two reserved words OBJECT and BODY are used to declare the relations (*Name, Secretary,* and *Manager*). OBJECT considers as a class that contains the declaration of the attributes of the relations *Name, Secretary,* and *Manager.* BODY considers as a relevant container that contains methods and constraints. Class *Name* contains three attributes *(First_name, Last_name,* and *Middle_init)* and two methods, in our example we just assigned initial values but it could be function or procedure. Class *Secretary* contains the attributes *(Classes, Birthday, Agent, S_Name.First_name, S_Name.Last_name,* and *S_Name.Middle_init)* and a method *Details.* Class *Manager* contains the attributes *(Classes, Class_description, Class_date, M_Name.First_name, M_Name.Last_name,* and *M_Name.Middle_init).*

The previous code appears as an efficient way to declare different types of objects in OODB. The problem is not easy to detect the violation of the database and check the integrity of the data because it's very difficult to detect all constraints that appear as a result of composite inheritance. The integrity constraints in OODBSs are maintained by rolling back any transaction that produces an inconsistent state, or disallowing or modifying operations that may produce an inconsistent state. Maintaining the violation of constraints needs to know which constraint violates the database and what is the covered solution. So it is suggested that an efficient way to help in maintaining the constraints when any violation or inconvenient circumstances appears.

Our suggested model is illustrated in Figure 4, is says that to combine all related attributes together under the reserve word ATTRIBUTE, same thing with METHOD and CONSTRAINT in one class. We added another method called *Detection* to express the status of the database.

```
CLASS class_name
 ATTRIBUTE     {user defined attributes}
 METHOD        {user defined operations}
 CONSTRAINT    {user defined operations}
 Detection     {it will be hidden}
END CLASS
```

Figure 4. **The general structure of the model**

Figure 5 shows the grammar of the suggested model, notice here the reserved word *Detection* should be hidden from the user because it will be declared by the OODBMS automatically for every class when puts the class

declaration. An initial value zero will be assigned to the *Detection* and the OODBMS changes its value depending on the database status.

The access of the specifier CONSTRAINT is declared the relationships and the constraints on the attributes of a class or between classes when inherit a subclass from a superclass. So the method *Detection* will contain the code that refers to the constraints status. The initial value is proposed to be zero to say that is no violation, and if an unexpected error happens then a code should be assigned to the *Detection* method.

```
S            → <classes>
<classes>    → CLASS <name> <members>
<members>    → ATTRIBUTE <attributes> METHOD <methods> CONSTRAINT
                <constraints> | ATTRIBUTE <attributes> CONSTRAINT
                <constraints> | ATTRIBUTE <attributes> METHOD <methods> |
                ATTRIBUTE <attributes>
<attributes> → <attributes> <name> <data type> | <name> <data type>
<methods>    → <methods> <operations> | <operations>
<constraints>→ <constraints> <condition> | <constraints>
<data type>  → integer | char | date | <classes>
<name>       → set of alphabet characters
<operations> → functions
<condition>  → special rules
```

Figure 5. The grammar for the suggested model

By using the "Detection" the status of the constraints can be checked at any time. The constraints are separated from the methods and the attributes. The constraints will be maintained or at least a message can be displayed about any error may appear as a result of data violation, then gives the programmer the choice to modify the constraints or refine the class members to avoid the violation.

7. IMPLEMENTATION

Using our proposed model the relation will be recreated as shown in Figure 6 the relation has been created to replace the code that was shown earlier in Figure 3.

```
CLASS Name
 ATTRIBUTE:
        First_name char (15),
```

```
            Last_name char (15),
            Middle_init char (1);
 METHOD:
            First_name := NULL;
            Last_name := NULL;
END CLASS;

CLASS Secretary
 ATTRIBUTE:
            Classes integer,
            Birthday date,
            Agent integer,
            S_Name Name;
 METHOD:
            Details:= NULL;
 CONSTRAINT:
            PRIMARY KEY (Classes);
END CLASS;

CLASS Manager
 ATTRIBUTE:
            Classes integer,
            Class_description char (15),
            Class_date date,
            M_Name Name;
 CONSTRAINT:
            PRIMARY KEY (Classes);
END CLASS;

CALSS Employee
 ATTRIBUTE:
            Person_ID integer,
            Recruiting_agent integer,
            Classes Manager,
            Details  Secretary;
 CONSTRAINT:
            PRIMARY KEY (Person_ID),
            FOREIGN KEY (Classes)
END CLASS;
```

Figure 6. Classes using the proposed model

Referring to the example in Figure 6, two subclasses *Manager* and *Secretary* are inherited from the superclass *Name. Employee* inherited from *Manager* and *Secretary.* The composite inheritance for attributes *M_Name* and *S_Name* from class *Name* gave them the same data type of *Name.* The *First_name, Last_name,* and *Middle_init* are added to the classes *Manager* and *Secretary* as it is inherited from same class. The constraints are inherited too and collected to be under CONSTRAINT so by grouping them the conditions that have been grouped can be checked easily. This effective especially when the user declares two constraints which conflict with each other as shown in Figure 7.

"All students that getting average more than 80 should be given $10"
C1: All s in student (where s.average > 80)
Implies: (all s in s.student get $10)

"All students that have average more than 90 should be given $20"
C2: All s in student (where s.average > 90)
Implies: (all s in s.student get 20$)

Figure 7. ALICE constraints for student schema

Suppose that C1 is a constraint over class A and C2 is a constraint over class B, and class C inherits the constraints from A and B (multiple inheritance). Constraints violate if the students grade is 95 because his grad is grater than 80 as C1 and grater than 90 as C2 (no violation between the conditions) so the student may get $10 or $20 (incorrect result) as a prize? It depends on which constraint will be enforced. A special code will be assigned to the *Detection* method. Figure 8 shows the mechanism of getting the code, assume *r1* is the rule that will be checked when inheritance happens. The involved constraints will be grouped and checked, if conflict exists then *Detection* method will be assigned with detection code and message will be displayed. The detection code is needed to modify or maintain the constraint.

```
r1 IS
    EVENT:                when inherit any subclass
    CONDITION: conflict between constraints
    ACTION:    display a warning message
END OF RULE
```

Figure 8. ALICE rule example over inheritance

The model will be activated during the classes' creation. When inheriting class from other class, the model will work and collects the involved constraints then checks the violation wither exist or not, then informs the programmer with the detection code if violation exists by displaying a warning message. So it will ease to avoid any database crash or data corruption.

8. CONCLUSION

Typically object-oriented databases lack the capability for an ad-hoc declarative specification of maintaining the integrity constraints [3, 6]. A new model to check and maintain the violation or unexpected circumstances of the object-oriented database is proposed. The model depends on the Assertion Language for Integrity Constraint Expression (ALICE) rule language to find the suitable way to maintain the violation by designing an efficient structural model to create the relations and its constraints.

The model that has been developed to detect the constraints over the relations is presented, to ensure global declarative specification and consistency maintenance using IRules in object-oriented database environment by applying ALICE rule. Supporting integrity constraints in object-oriented database systems requires a high integration of the constraints with the rich concepts available.

With the rich semantics of object-oriented paradigm a lot of work remains to be done for future work. In particular, more optimization techniques can be developed for constraint compilation. Object-oriented databases made new challenges to semantic integrity especially to both constraint representation and constraint maintenance.

REFERENCES

[1] Ina Graham, *Object-Oriented Methods Principles & Practice.* England: Addison-Wesely, 2001.

[2] David W. Embley, *Object Database Development Concepts and Principles.* England: Addison-Wesely, 1998.

[3] Bindu R. Rao, *Object-Oriented Database Technology, Applications, and Products.* US: McGraw-Hill, 1994.

[4] Setrang Khoshafian, *Object-Oriented Databases.* New York: John Wiley & SonsInc, 1993.

[5] Urban. ALICE: An Assertion Language for Integrity Constraint Expression. Proceedings of the Thirteenth Conference on Computer Software and Applications, 1989; pp. 292-299.

[6] S. Ceri and J. Widom. Deriving Production Rules for Constraint Maintenance. Proc. 16th Int'l Conference Very Large Data Bases, 1990; pp. 566-577.

[7] Susan D. Urbana and Anne M. Wang. The Design of a Constraint/Rule Language for an Object-Oriented Data Model. Elsevier Science Inc., J.system software 28, 1995; pp. 203-224.

[8] Urban, Karadimce, and Nannapaneni. The Implementation and Evaluation of Integrity Maintenance Rules in an Object-Oriented Database. Proceedings of the Eighth International Conference on Data Engineering, 1992; pp. 656-572.

[9] Urban. Desiderio. CONTEXT: A Constraint Explanation Tool. Data and Knowledge Engineering, North-Holtand, 1992; pp. 153-183.

[10] Michael Sipser, *Introduction to the Theory of Computation.* PWS Publishing Company, 1997.

[11] Ying Jin, Amy Sundermier, and Suzanne W.Dietrich. An Execution and Transaction Model for Active, Rules-Based Component Integration Middleware. Springer-Verlag Berlin Heidelberg 2002; pp. 403-417