Designing a New Assertion Constraints Model for Mobile Databases

http://dx.doi.org/10.3991/ijim.v6i2.1836

Belal Zaqaibeh[1], Firas Albalas[1], A. W. Awajan[2]
[1] Jadara University, Irbid, Jordan
2Al-Balqa' Applied University, Jordan

***Abstract*—In this paper a new assertion constraint model is proposed and implemented. The model is designed to enforce and maintain the integrity constraints in mobile databases and object data model environments. The object assertion model for integrity constraints is used to create classes and collected attributes and their constraints that are derived from multiple compositions and inheritance hierarchies. Also it has a compile-time model which keeps the derivation path along with the attributes' relationships. Furthermore, the run-time model enforces integrity constraints and the logical integrity constraints during the run-time. And a new technique is designed to check the object metadata to detect the object violation before it occurs. However, the model is implemented and tested over set of definitions that check attribute values validity and objects for object-oriented data model and mobile databases.**

***Index Terms*—Constraints violation, Mobile database, Mobile database model, Object data model.**

I. INTRODUCTION

Recently, there has been an increasing interest in mobile computing due to the rapid developments in wireless communication and portable computing technologies [1]. A general architecture of a Mobile Database (MDB) environment consists of base stations and mobile hosts [13]. The mobile host is the mobile component that moves from one cell to another, and communicates with the base stations through wireless networks where MDBs are specialized class of distributed systems [10]. Due to limited storage capabilities [1], the mobile host is not capable of storing all data items in the network, thus it must share some data item with a database in the used network.

In Object Data Models (ODMs) regardless whether it is Object-Oriented Databases (OODBs) or MDBs, data accuracy, consistency, and integrity in are extremely important for developers and users. Checking and maintaining the Integrity Constraints (ICs) is a fundamental problem [5], [15]. ICs are conditions that data within a database must satisfy. Checking for ICs to maintain the consistent state of MDBs is an important issue that needs to be addressed [13].

This paper presents our contribution for this research, in which it clarifies the proposed model properties and specifications including Object Assertion Language for ICs (OALIC), Object Metadata (OMD), and Detection Method (DM). Furthermore, this paper presents the enforcement and maintenance technique of the Compile-Time Model (CTM) for Structural ICs (SIC) [2] and also the Run-Time Model (RTM) for Logical ICs (LIC).

This paper is organized as follows. Section II presents the groundwork of our research. Section III presents the related work. The proposed model framework is presented in section IV. Also section V explores the OALIC and its structure format and grammar. Consequently, the model components are presented in section VI. Subsequently, section VI.A presents the CTM for SICs, while the RTM for LICs is presented in section VI.B and the DM in sections VI.C also section VI.D presents the OMD features, which includes three classes that are the constraint optimization, constraint knowledge, and knowledge base. The enforcing and maintaining of ICs is presented in section VII. Naturally, we ended this paper by a conclusion and future work in section VIII.

II. PRELIMINARIES

In advanced office automation systems the MDB and OODB are used to handle hypermedia data. Image processing and designing systems use ODM technologies for ease of use. All of these applications are characterized by manage complex and highly interrelated information, which is the strength of ODM. The increased emphasis on process integration is a driving force for the adoption of OODB systems [2], for example, the computer aided design (CAD) area is focusing heavily on using OODB technology as the process integration framework. Clearly, relational database technology has failed to handle the needs of complex information systems [3].

The MDB system is a distributed system based on client-server diagram [6]. Checking ICs in MDB systems is more complex compared to conventional database recovery because of an unlimited geographical mobility of mobile hosts. MDB uses database dependent information such as metadata or use specific functions of database server such as trigger and time stamp. These constraints are critical weak points in ubiquitous environment because various applications are running in various devices in ubiquitous environment [13].

Integrity maintenance or constraint enforcement is a set of activities that keeping databases in a consistent state [11]. ICs in OODBs are maintained either by rolling back transactions that produce an inconsistent state, or by disallowing operations that may produce an inconsistent state for the constraints [2], [4]. Existing ODM management systems lack the capability for an ad-hoc declarative specification of maintaining the ICs. An alternative approach is to provide automatic detection of inconsistent states. For each constraint, a rule is used to detect constraints violation and to initiate database operations to restore consistency.

Some ICs are represented and maintained naturally in OODBs by capturing the violation using the type system and the class hierarchy. Checking the ICs in OODBs is a fundamental dilemma in database design [3], because current OODB management systems lack the capability of an ad-hoc declarative specification of maintaining ICs that appear as a result of composition, inheritance, and association hierarchies. The constraints must be maintained in the forward direction along the class composition hierarchy as well as in the backward direction. The model can represent ICs and their relationships over the composition and inheritance hierarchies [5].

The automated verification of ICs and their enforcement provided by current OODB management system is limited [5]. A database state is said to be consistent if the database satisfies a set of statements, called semantic ICs [8]. Handling semantic ICs is an essential premise to manage semantically rich data [2], [3]. In addition, handling ICs is an essential premise to managing semantically rich data [3].

The new proposed model can handle structural integrity (constraint base). Typically, we consider the ODM as the underlying data model where it includes MDB and OODB. The maintenance methodology here depends on fixed values domain and attributes domain.

The fixed values domain is a finite (e.g., set of integers between 2 and 7) or an infinite (e.g., set of characters) set of values, and the domain of an attribute is a finite set that includes data in a particular object. Attributes are members in a class and they represent data components that make up the content of a class. The term class refers to a collection of all objects with the same internal structure (attributes and methods) [7]. ODM is based on the concept of a class. Within the OODB management system, the class construct is normally used to define the database schema. The OODB management system schema identifies all the objects stored within the database, these objects are known as instances of the class. These instances of class carry once and for all, the data values of class attributes.

Two steps must be taken when a user request is submitted. First, all constraints that may be violated by any transaction must be specified. Specifically, we should check if each constraint may be violated. Second, if IC would be violated and will be in inconsistent state for the OODB, then a proper action must be taken (e.g., aborting or modifying the current user request) such that it will be true in the new ODM state.

III. RELATED WORK

A framework is proposed by Dzolkhifli [13] for caching relevant data items needed during the process of checking ICs of MDBs. Dzolkhifli has analyzed the relationships among the integrity tests to be evaluated for a given update operation. This improves the checking mechanism by preventing delays during the process of checking constraints and performing the update, this model speeds up the checking process.

The proper handling of ICs is essential to any data storage and management. Handling ICs is an essential premise to managing semantically rich data [3]. In OODBs, checking the ICs is a fundamental problem in the database design [3]. The automated verification of constraints and their enforcement provided by current OODB management systems is limited [3]. Many researchers have studied the problem of enforcing ICs in MDBs and OODBs and many different approaches have been proposed, but none of the approaches has addressed the issue of maintaining User Defined Constraints (UDCs) in composition and inheritance hierarchies.

A set of security vulnerabilities is identified by Ghorbanzadeh [10] on MDB to apply appropriate technique to decrease side affects of MDB security by tacking into account the ICs [10]. Also, an architectural model is presented by Abiona [21] for wireless peer-to-peer file sharing system for ubiquitous mobile devices. The proposed model is based on a hybrid or semi centralized architecture with the central database server acting as an interface between the mobile devices.

Choi has proposed an algorithm [20] which is called the synchronization algorithms based on message digest in order to facilitate data synchronization between a server-side database and a MDB. The OODB management systems do not have adequate support for certain types of constraints especially the ones defined in a class composition and inherence hierarchies [3], [9], [17], [18]. The ICs must be maintained in the backward direction along the class hierarchies as well as in the forward direction. It seems to be no obstacles in extending the proposed model to deal with constraints.

Maintaining constraints is not an easy process in inter-constraint. The inter-constraint maintenance problem and the contradiction or lack of proper functionality of a set of constraints is addressed in [3]. Also, in [19] there is an issue of commercial semantic databases that extensively supports structural integrity enforcement and arbitrary constraint checking. Other work in [2], constraints have been done from the aspect of constraint satisfaction and constraint logic programming languages, where the emphasis on using constraints propagation.

IV. THE MODEL FRAMEWORK

The proposed model framework is consisted of five components namely: ODM, OMD, DM, rules, and applications as shown in Figure 1. To have a complete working model, a modeling language is required. Therefore, The OALIC which is illustrated in the next section is used to create the classes and their attributes, methods, and constraints. The model gathers all attributes and constraints, identifies the relationships, optimizes constraints, and stores them in the OMD.

The attributes and methods represent entities and their behaviors. A constraint can be defined on attributes. The participating objects are represented by translating UDCs into rules and storing them in the OMD. The user does not

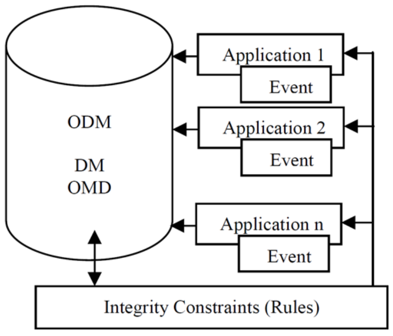

Figure 1. The Model Framework

have to specify detailed execution procedure for constraint checking and propagation in the database. The constraints are translated into rules and relationships regardless whether the propagated constraints are derived from composition or inheritance hierarchies.

The DM is used to update and retrieve information from the OMD. The OMD is used to manage constraints and also to store the attributes and their paths, constraint knowledge, and constraint base. Furthermore, the OMD provides operations that eliminate conflicts of constraints.

All the operations are based on the OMD contents. The DM checks the constraints in the OMD when a user request is received from applications. It is not easy to enforce the ICs when composition or inheritance hierarchies exist. This is due to the fact that detecting constraints that appear as a result of inheritance and composition hierarchies requires backward and forward detection method. The DM reads the OMD, finds the involved constraints, checks the involved attributes that are needed to be modified during the objects creations, and also checks the objects.

When a transaction is received, the DM gets all information about the involved attributes and their constraints, and verifies the new changes that may happen due to the user request. If there is no violation then the DM gets the new changes and updates the OMD. But if the transaction causes inconsistent state, the DM gets the required information about the violation from OMD and stores them in its variables then sends them to the databases management system to abort the running transaction. Before the violated constraints are maintained, the model recognizes which constraints may violate the database and what is the repairable action. Therefore, the limitations of Do's approach [9] are overcome by collecting the constraints information in the OMD and call only the involved constraints when an event occurs. A practical working technique in maintaining the constraints when any violation or unexpected circumstances occur is already implemented and tested.

V. THE OALIC

The OALIC is the assertion language that is proposed to handle classes and their attributes and constraints. The OALIC is designed to simplify constraints to any ODM.

A. *The Structure Format of OALIC*

The general structure of the OALIC is illustrated in Figure 2. All attributes are gathered under the specifier ATTRIBUTE, behaviors or methods under the specifier METHOD, and CONSTRAINT is added to gather constraints [15]. Also a new method called DM is introduced to express the status of the constraints.

The user can manipulate (insert, delete, or modify) attributes, methods, and constraints. But he/she cannot see or deal with the DM directly. The DM will be hidden from the user because it supports the ODM management system with the knowledge about the constraints, and only the management system can read/access its value.

```
CLASS class_name
   ATTRIBUTE      //user defined attributes
   METHOD         // user defined behaviors
   CONSTRAINT     // user defined constraints
   DM             // hidden detection method
END CLASS
```

Figure 2. The General Structure of OALIC Format

The model has been designed to maintain redundant (subset), inconsistent (conflict), and duplicate constraints, also to enforce ICs and keep the database in a consistent-state. The enforcement technique keeps the consistency among the constraints, so if a violation is expected to be occurred, several actions will be done as follows:

- Sending the current user requests and the constraint derivation path to the maintenance technique.
- For each user request, the DM checks the OMD and assigns the new values to the DM variables.
- The DM will be verified then the dependences among constraints will be specified.
- If the constraints cannot be maintained then the error handler technique keeps the violation path and type. Moreover, the user request is aborted.

B. *The OALIC Grammar*

The EBNF grammar of OALIC as per the ISO/IEC rules format [12] is shown in Figure 3.

```
<start>         ::= class <identifier>
< identifier>   ::= <classname> <description>
<classname>     ::= <name>
<description>   ::= [: (<classname> {,<classname>})] <member>
<member>        ::= attribute <attmember>
<attmember>     ::= {<field>}+ [<funmember>]
<funmember>     ::= method <mthmember>
<mthmember>     ::= {<operation> }+ [<conmember>]
<operation>     ::= <funheader> <oprbody>
<funheader>     ::=  <name> ( {<parameter>} ) <datatype>
<parameter>     ::= <variable> {, <variable>}<datatype>
<oprbody>       ::= begin <usercode> end ;
<usercode>      ::= <assign> | <condition> | <loops>
<condition>     ::= if <cond> then <statement> else <statement>
<cond>          ::= <numeric> | <variable> | <exprbinary>
<conmember>     ::= constraint {<rule> ; }+
< field>        ::= <attname> <datatype>;
<attname>       ::= <name>
<exprbinary>    ::= <expr> <operator> <exprbinary> | <expr>
                    <oprbool> <exprbinary> | <expr>
<loops>         ::= <forloop> | <whileloop> | <repeatloop>
<forloop>       ::= for <expr> (to | downto) <expr> do <body>
<whileloop>     ::= while ( <cond>) do <body>
<repeatloop>    ::= repeat <body> until <cond>
<body>          ::= begin <statement> end;
<statement>     ::= <usercode>
<assign>        ::= <variable> := <expr>
<expr>          ::= <term> { (+ | −) <term>}
<term>          ::= <factor> { (* | /) <factor>}
<factor>        ::= <expr  | <variable> | <numeric>
<datatype>      ::= <basictype> | <collection> | <classname>
<basictype>     ::= integer | real | boolean | char | string | date
<attpath>       ::= <variable> {. <classname>} . <attname>
<funpath>       ::= <variable> {. <classname>} . <funname>
<rule>          ::= {<ruleobj>}+ [<ruleext>]
<ruleobj>       ::= <intraobj> | <interobj>
<ruleext>       ::= (<operator> | <oprbool>) <rule>
<interobj>      ::= <operand> <operator> (<attpath> | funpath) |
                    <attpath> <operator>  | <intercmlx>
<intercmlx>     ::= <interobj> <oprbool> <intercmlx> | <interobj>
<intraobj>      ::= <operand> <operator> <intraobj> | <operand>
<intracmlx>     ::= <intraobj> <oprbool> <intracmlx> | <intraobj>
<operand>       ::= <variable> | <attname> | <funname>
<variable>      ::= <letter> { (<letter> | <digit> | _ ) }
<collection>    ::= (<settype> | <bagtype> | <listtype> | <ar-
                    raytype>)  <basictype>
<name>          ::= <variable>
<arraytype>     ::= array (<numeric> [,<numeric>] )
<numeric>       ::= {<digit>}+
<operator>      ::= > | < | = | >= | <= | <>
<oprbool>       ::= and | or | xor
```

Figure 3. The EBNF Grammar for the OALIC

Typically, objects are declared from classes. A class has a name and a set of members. The inherited members from the superclass become members in the subclass. Members in a class can be attributes, methods, or constraints. Each attribute has a data type and domain. The data type can be basic (e.g., integer, real, etc.) or structural data type (e.g., set, bag, or class). The attribute domain can be static (e.g. 1, 2, 7, etc.) or dynamic (e.g., attribute values).

The constraints are the conditions that control attributes values. The constraint operands that enforce attribute values in OALIC are constant value, literal, attributes, expression, and aggregate function. Once a class is created, all members are gathered and their relationships are specified. The idea behind gathering class members is to find the relationships, specify the dependences, and keep the derivation path.

Typically, all relationships and the derivation path are kept for each class member. Therefore, all constraints (constraint base) and relationships (constraint knowledge) will be collected, analyzed, optimized, and stored in the OMD. The hierarchy model is used to keep the derivation path, constraint base, and constraint knowledge.

VI. tTHE MODEL COMPONENTS

The model has the following main components as Figure 4 shows:

- CTM: During the compile-time, the CTM enforces and maintains SICs where it is performed only once, and the user who interacts with the system and supplies it with added information, is referred to as the constraint designer.
- RTM: During the run-time, the RTM enforces LICs where it is performed whenever an update is submitted for processing and the user who uses the real system, is referred to as the end-user.
- OMD: It is an object data structure containing a record for each constraint and attribute, where it allows to find the record for each identifier quickly and to store and retrieve data from that record quickly too.
- DM: It is the interface between OMD and the object data model, where it reads the OMD and allows or disallows transactions to be performed.

A. *The CTM*

The CTM which appears in the upper part in Figure 4, is responsible for enforcing and maintaining SICs [5].

The CTM starts from the user interface, which it forms the interactive interface and handles the communication between the constraint analyzer and the constraint designer. The constraint analyzer is responsible for analyzing constraints in order to discover its phrase structure, and distributing the analyzed constraints to the constraint parser in order to check the constraint syntax. The constraint parser takes the stream of tokens and uses them to construct hierarchical structures called parse trees.

The parse trees represent the systematic structure of the constraint. The error handler is responsible for handling errors as it receives the invalid constraints and constraints that cannot be maintained then reports the violation knowledge. However, after detecting an error, the present phase somehow deals with that error. The constraint checker is responsible for checking the constraints and recognizing the accepted and unaccepted constraints. Definition 1 illustrates the constraints status.

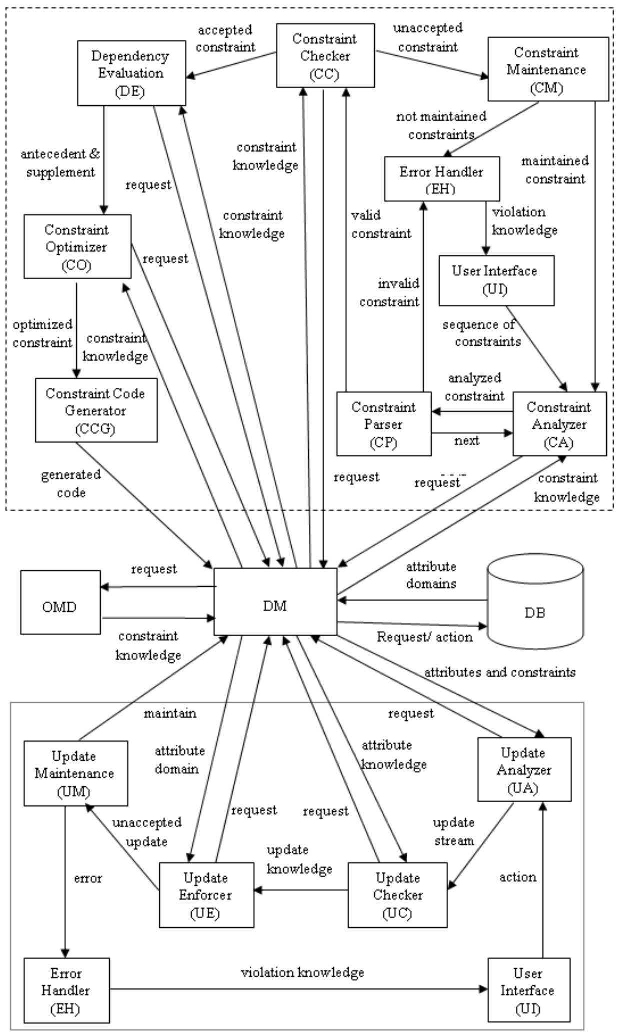

Figure 4. The Model Architecture

Definition 1: Let ς be a constraint that is derived from a set of classes (*cls*). The ς might be an accepted constraint ξ, or an unaccepted constraint ζ (redundant, inconsistent, or duplicate constraint).

Definition 2: Let ζ_r be a redundant constraint, ζ_i be an inconsistent constraint, and ζ_d be a duplicate constraint. Therefore, if ζ exists in any form of (ζ_r, ζ_i, or ζ_d) then ζ will not be accepted until it is maintained.

The constraint maintenance is responsible for maintaining ζ_r, ζ_i, and ζ_d constraints. The constraint maintenance has two types of functions which are maintaining the SICs and the LICs. The dependency evaluation is responsible for specifying the constraints' domains, Antecedent Constraint (AC) (inherited constraint), and Supplement Constraint (SC) (derived constraint), then sends them to the constraint optimizer. The constraint optimizer is responsible for reducing coupling, as it eliminates the unnecessary relationships among the constraints. Therefore, this reduces the execution time, increases the execution speed, increases the compilation time, and reduces the compilation speed. Subsequently, the constraint optimizer increases the model efficiency because of the fact that, execution speed is more important than compilation speed [14].

B. The RTM

The RTM is responsible for enforcing the ICs, verifying transactions (inserting, updating, and deleting objects), checking constraint domains, and maintaining the unaccepted user request [15], [16], and its architecture is shown in the lower part in Figure 4.

The RTM communicates with the DM to get the constraints and attributes information. However, all transactions must remain the database in a consistent state. In the RTM the user interface forms the interactive interface, which handles the dialogue between RTM and its users. Users may delete or insert objects and the RTM handles their actions. The update analyzer uses the knowledge about the constraints that are provided by the DM and maps each update request into a set of domains then sends them to the update checker with the involved attributes and constraints. The update maintenance is responsible for maintaining the unaccepted user requests, after detecting an error the present phase must in somehow deal with that error.

The DM has several functions that depend on the connection phase. Subsequently, the DM receives update analyzer requests and accesses the OMD to get the attributes and constraint knowledge that is stored by the CTM during the compile-time [5] then sends them to the update analyzer. Moreover, the DM receives the actions from the UM if a user request needs to be maintained whereas the UM does the maintenance.

The RTM receives user requests and then analyzes them to determine the action type with the help of the DM by sending requests and receiving knowledge about the involved attributes and constraints. Moreover, after all required requests information is collected the UA sends streams for checking purpose.

C. The DM

The DM is the interface between OMD and the database management system, and also an intermediate function between CTM and RTM. The DM reads the OMD and allows or disallows transactions to be performed. The DM is an overloaded method that can access and modify the OMD[15]. The DM is designed for constraint validation checking purpose. Therefore, the DM has two functions that are differentiated from each other by their arguments as follows:

- DM (CID, AID, RCID, RAID, {AC}, {UDC}, {SC})
- DM (CID, AID)

The Constraint ID (CID) and Attribute ID (AID) are the composite key for reaching the information about all attributes in the OMD. This information includes constraints base, derivation path, domains, derived attributes, and superclasses. The CID is a unique ID, this means the CID cannot be repeated even if an object is deleted and then declared. The AID represents the ID for an attribute in a particular object, where the ID is unique under the class level; this means the AID can be repeated under different classes. The RCID represents the ID for the superclass if the attribute is derived from inheritance or composition hierarchies. The RAID represents the ID of an attribute when the current attribute is derived from other attribute.

D. The OMD

The OMD is the constraints map. It is responsible for building the specific knowledge base of the constraints of DBs and is built once by the CTM [5]. The OMD is a data structure containing a record for each constraint and attribute. The data structure allows to find the record for each identifier quickly and to store or retrieve data from that record immediately too. Each attribute has a domain, which is the valid value that can be stored in a particular attribute.

A domain attribute is the range of its data type or set of values that are controlled by constraints in different ways like constant values, literals, attributes or aggregate functions. An essential step is that, simplifying the constraints in domains, this means determine the attribute domain by its data type and constraint. The attribute domain controls the attribute values in the OMD. The OMD stores all classes, attributes, constraints, and their relationships.

The OMD has been designed to manage constraints that are in independent, inherited, composed, and associated classes. The OMD consists of three classes as shown in Figure 5, namely: OMD Constraint Optimization (OMD^{CO}), OMD Constraint Knowledge (OMD^{CK}), and OMD Knowledge Base (OMD^{KB}). The OMD classes are used to describe objects structure. The OMD has all the required information about the constraint base and constraint knowledge.

The following sections describe the OMD classes, where these classes are connected with association and composition relationships.

E. Constraint Optimization Class

The OMD^{CO} is the constraint optimization class that optimizes the constraints and the domains. The OMD^{CO} includes the constraints, domains and Domain ID (DID). The DID is the hashing key that creates a method for searching as opposed to simply scanning a large data with all the nodes. In addition to, it makes addition and removal of nodes more efficient. Furthermore, the DID is associated with the OMD^{CK} class with M:N as shown in Figure 5. Thus, the DIDs indicate domains for associated attributes.

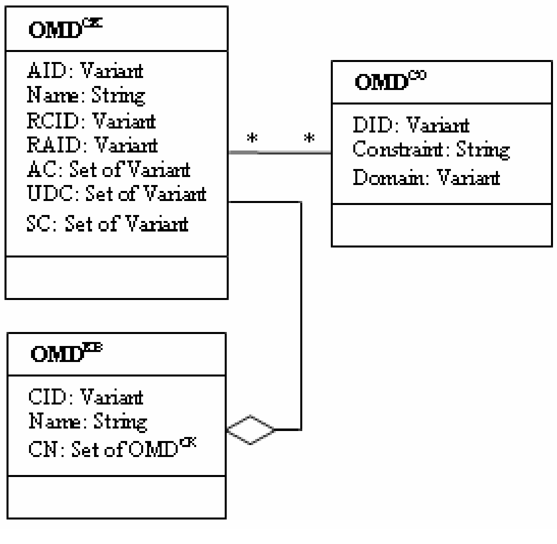

Figure 5. The OMD Structure

For the model optimization technique, the Directed Acyclic Graph (DAG) is extended and an Optimization Method (OM) is developed to support dynamic values, objects, constraints, and domains. The advantage of DAG is that it avoids redundant of sub-trees. Typically, a constraint can be merged into a single domain. While DAG avoids redundant code, it can be inefficient and problematic later on when changing values from time to time (dynamic values). In order to implement a DAG, usually the nodes are stored in an array and searched when a new node is to be created. The OM is designed to handle dynamic values and overcome the DAG drawback.

Definition 3: Let C_i be a constraint in $C_1, C_2, \ldots, C_n$, D_i a domain in $D_1, D_2, \ldots, D_n$, and $dom(C_i)$ is the method that generates D_i of C_i. Typically,

$D_i = dom(C_i)$

If $dom(C_i) = D_i$ and $dom(C_j) = D_j$ where $D_i = D_j$

$\Rightarrow dom(C_i) = dom(C_j) = D_i$

$\Rightarrow D_j$ will be eliminated

The constraints and their domains will be optimized in the OMD^{CO} during the CTM using the Algorithm 1:

Algorithm 1: Optimization

```
Input:    Set of ξ {ξ1, ξ2, … ξn }
Output:   OMD
Steps:
1    Start
2    SofC ← ∅
3    While i ≤ n Do  //n: number of ς in cls
4        SofC[i] ← ξi  //extract constraints from cls
5        Increment i
6    End while
7    For SofC[i] = 1 to n
8        For OMD(j) ← 1 to d Do //d: number of domains in OMD^CO
9            If dom(SofC[i]) = OMD.dom(ξj) Then
10               Find the equivalent domains
11               Eliminated dom(ξ)  //Definition 3
12           Else
13               OMD^CO ←dom(ξ)
14               OMD^CO ← DID           // continual DID
15           End If
16       End For   // j
17   End For        //i
18   End
```

F. *Constraint Knowledge Class*

The OMD^{CK} is the class for collecting the attributes, constraints, and domains knowledge. For each attribute in the class there is a unique identifier called AID as shown in Figure 6. Thus, the DM can access and control any attribute using AID. The model can enforce attributes integrity in classes that are result of association, composition, and inheritance hierarchies. However, to keep the derivation path the OMD^{CK} keeps the RCID and the RAID that are derived into the present class.

Users can declare constraints, which are called UDCs (e.g., parent.age > 16). Typically, a UDC may depend on an ACs that are derived from superclasses, associated, or composed classes. Thus, the ACs must be verified before the UDCs verification (e.g., child.age < parent.age). Subsequently, a SCs are constraints that depend on UDCs, thus, the SCs must be verified after the UDCs verification.

Since the OMD^{CK} has an association relationship with the OMD^{CO}, the AC, UDC, and SCs take their values from the OMD^{CO}. The AC and UDC are declared from sequence data type of DID, and SC from sequence data type of DM. Whereas the AC represents the DID of the domain

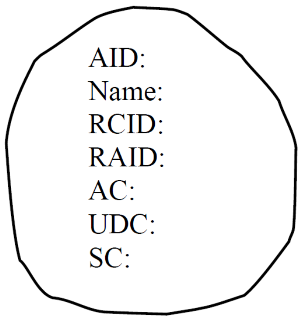

Figure 6. An Instance of OMD^{CK}

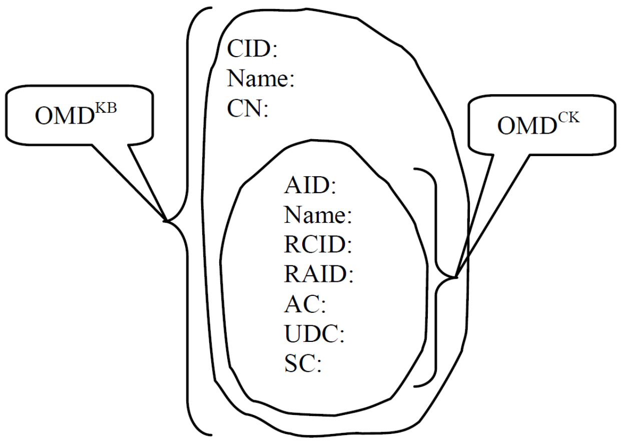

Figure 7. An Instance of OMD^{KB}

of dependent attributes, so before any update is the domain in the OMD^{CO} must be satisfied. The domains of the DIDs in the UDC must be satisfied too to remain a consistent state for the database. Accordingly, to prevent a violation that may occur in other attributes, the DIDs in the SC must be checked and satisfied. If a violation occurs in any of AC, UDC, or SC the database management system will abort the current user request.

Typically, if there is a constraint on a particular attribute, the DID of the constraint will be stored under the UDC for that attribute. Subsequently, if there is a set of constraints on a particular attribute, the DIDs will be stored under the UDC for that attribute. Furthermore, if a constraint is inherited by a particular attribute as a result of inheritance hierarchy, the DID of that constraint will be stored in the AC for that particular attribute. Also the RCID and RAID indicate the derivation path. The SC stores the DMs that depend on the present attribute.

G. *Knowledge Base Class*

The OMD^{KB} is the structure for the OMD object. The OMD^{KB} includes knowledge about all attributes and their relationships, constraints, and domains. Each class has a unique internal identifier called Class ID (CID) and a unique name. The OMD^{KB} composes a sequence of OMD^{CK} in the CN attribute. The OMD is the instance of OMD^{KB} class as shown in Figure 7.

In the OMD the OMD^{CK} is a composed object for the CN attribute. Each object in OMD represents class knowledge. So to enforce a particular constraint we need to read only the related object for that constraint and only the involved constraint will be verified, so this reduces the execution time and avoids multitasking, accordingly, this increases the model efficiency.

VII. ENFORCING AND MAINTAINING ICS

Constraints can enforce a finite set of values for attributes in one class as well in many inherited and composed classes. Relationships between classes inherit members from superclasses to subclasses. In some cases, conflict among constraints may occur. The CTM can enforce and maintain ICs that are propagated from composition and inheritance. Furthermore, CTM can also enforce and maintain constraints that are derived from mixed of such relationships.

Typically, detecting and checking this type of constraints is very important since the derivation path in composition hierarchies cannot be detected. Enforcing ICs in RTM occurs during the run-time, so the maintenance of RTM is required whenever events are submitted. Therefore, there are two general steps to be performed. First, all constraints that would be violated must be found. Second, determining what actions must be taken.

Generally, an object may have a set of attributes, so when inserting a new object, all attributes and constraints will be verified. If a constraint is not satisfied then this will violate the database.

As mentioned earlier, the OMD will be generated in the CTM. Therefore, the DM will be called to read the OMD and verify whether the new update will violate the database or not. The DM will call each attribute in the following format:

- DM(CID, AID, RCID, RAID, AC, UDC, SC)

Then verifies whether the values are accepted in the intended attributes or not. The DM verifies the AC, UDC, and SC for each called attribute. The idea is to instantiate the relevant constraint with the object to be inserted, updated, or deleted. Then the processes are simplified by eliminating unnecessary comparisons. The simplified form of the constraint is evaluated before an object is inserted to the database. The process before enforcing LICs in RTM is to create the OMD that includes all the knowledge about classes and their members. Since we deal with UDCs regardless whether classes are designed in a good or bad design, all constraints and domains are verified, optimized, and collected in the OMD.

- Inserting Object: When inserting a new object, all constraints in OMD that are related to that object must be checked to verify the new data state.
- Deleting Object: Deleting object from independent classes (intra-class constraints) does not require verifications for any constraint. In the contrary, deleting object from dependent classes (classes with composition, inheritance, or association relationships) requires verifying the SCs only in the deleted objects and also the ACs, UDCs, and SCs in the associated, inherited or composed objects.
- Updating Object: It requires keeping the current database state D until verifying the ICs in D^{+}.

If the updating request is rejected then the cause of violation and its path will be known. And this is a clear advantage of the model, as the current object-oriented applications do not have the ability to support the violation path.

At this point, we concentrate on the ICs and data integration for satisfying a set of rules. We extend the formal ODM with standard operators by including two aggregation operands. One of the most significant problems is the incorporation of UDCs with the composition and inheritance mechanisms in the ODM. We have overcome this problem by layering the model over the ODM. This approach has several benefits:

- Represent complex relationships and relationships that are propagated from association, composition, and inheritance hierarchies.
- ICs can constrain the action of computationally methods.
- ICs can be applied to arbitrarily complex objects including hierarchy structures.
- ICs are at a higher level of abstraction and thus easier for users to read and write.
- Support multiple processors to maintain the constraints simultaneously when sets of objects or constraints of different relationships are completely independent from each other.
- Can be integrated with any existing or specialized constraint services.
- The violated constraints are to be maintained automatically by the maintenance technique.

VIII. CONCLUSION AND FUTURE WORK

The proposed model has made a big challenge in the ODM environment as it can represent constraints and complex relationships among attributes and classes that are derived from composition and inheritance hierarchies, whereas the current ODMs are deficient in such properties. The model is implemented and tested over MDBs and OODBs.

Since the ICs are conditions that data within a database must satisfy, so database must have a set of activities that enforce integrity and maintain constraints to keep the database in a consistent state. This paper has shown the proposed model properties and specifications including CTM, RTM, and OM. The model has made a big challenge in the ODM environment as it can represent constraints and complex relationships among attributes and classes that are derived from composition and inheritance hierarchies. The model is able for enforcing and maintaining ICs in SICs by CTM and LICs by RTM.

The OALIC grammar facilitate the usage of the model and make it competent to be used by any existing ODM. Also it can enforce the ICs for constraints with two operands that are not supported by the current MDBs. The OMD has three classes namely: OMD^{CO}, OMD^{CK}, and OMD^{KB}, to keep track the constraint paths in the backward direction as well in the forward directions. The OMD^{CO} has a special new technique that is built based on DAG to reduce coupling among attribute relationships and domains. The OMD^{CK} keeps constraint knowledge to ease accessing them. Furthermore, the OMD^{KB} is designed to include knowledge about all attributes and their relationships, constraints, and domains by composing OMD^{CK} that are associated with OMD^{CO}.

The CTM is implemented and set of definitions are supported for checking whether a constraint is valid or invalid, and also checking redundant, inconsistent, and duplicate constraints. Furthermore, the RTM is implemented too and clarified with its properties, specifications, and architecture. A set of definitions is supported for checking attribute values validity, database consistency, and also a method for verifying attribute values when inserting, deleting, and updating objects.

This model can be improved by developing more optimization techniques for constraint compilation. ODMs face new challenges to semantic integrity especially to both constraint representation and constraint maintenance. More work can be done when copying an object of a superclass to another object of a subclass and vise versa. For such problem down-casting and slicing must be taken in account. Moreover, when a multiple inheritance occurs and the same attribute name existed in more than one superclass, then a virtual class is needed.

REFERENCES

[1] Dzolkhifli Z., Ibrahim H., and Affendey L., "Analyzing integrity tests for data caching in mobile databases", ICDCIT, pp. 157-165, Springer-Verlag, 2008.

[2] Zaqaibeh B., and Al-Daoud E., "The Constraints of Object-Oriented Databases", International Journal of Open Problems in Computer Science and Mathematics (IJOPCM), Vol. 1, No. 1, 2008.

[3] Formica A., "Finite Satisfiability of Integrity Constraints in Object-Oriented Database Schemas". The IEEE Transactions on Knowledge and Data Engineering, Vol. 14, No. 1, pp. 123-139, 2002. http://dx.doi.org/10.1109/69.979977

[4] Date C. J. An Introduction to Database Systems. 8th edition, Addison Wesley, 2004.

[5] Ibrahim H., Zaqaibeh B., Mamat A., and Sulaiman M., "Enforcing and Maintaining Constraints Base during the Compile-Time", the World Scientific and Engineering Academy and Society (WSEAS) Transactions on Computers, Vol. 6, No. 2, pp: 373-379, 2007.

[6] Mahmoodi M., Baraani A., Khayyambashi M., "Recovery Time Improvement in the Mobile Database Systems", International Conference on Signal Processing Systems, pp: 688-692, 2009.

[7] Sarhan A., "A New Allocation Technique for Methods and Attributes in Distributed Object-Oriented Databases Using Genetic Algorithms", the International Arab Journal of Information Technology (IAJIT), Vol. 6, No. 1, 2009.

[8] Ibrahim H., "A Strategy for Semantic Integrity Checking in Distributed Databases", Proceedings of International Conference on Parallel and Distributed Systems, pp: 139-144, 2002.

[9] Do N. C., Choi I. J., and Jang M., "A Structure-Oriented Data Representation of Engineering Changes for Supporting Integrity Constraints". The International Journal of Advanced Manufacturing Technology, Vol. 20, No. 8, pp. 564-570, 2002. http://dx.doi.org/10.1007/s001700200192

[10] Ghorbanzadeh, P.; Shaddeli, A.; Malekzadeh, R.; Jahanbakhsh, Z.;, "A survey of mobile database security threats and solutions for it", 3rd International Conference on Information Sciences and Interaction Sciences (ICIS), pp: 676-682, 2010.

[11] Ibrahim, H., "Checking Integrity Constraints – How it Differs in Centralized, Distributed and Parallel Databases", Proceedings of the 17th International Conference on Database and Expert Systems, pp: 563-568, 2006.

[12] ISO, 1996. ISO/IEC 14977: 1996 (E).

[13] Dzolkhifli, Z.; Ibrahim, H.; Affendey, L.S.; Madiraju, P. "A framework for caching relevant data items for checking integrity constraints of mobile database", International Journal of Interactive Mobile Technologies (iJIM), Vol.3, No.2; pp: 18, 2009.

[14] Aho A., Sethi R., and Ullman. "Compilers Principles, Techniques, and Tools". Addison Wesley, 1986.

[15] Zaqaibeh B., Ibrahim H., Mamat A., and Sulaiman M., "Enforcing User-Defined Constraints during the Run-Time in OODB", the International Arab Journal of Information Technology (IAJIT), Vol. 5, No. 4, 2008.

[16] Zaqaibeh B., Al-Hanandeh F., and Al-Daoud E., "Development of a Run-Time Model in OODB". Proceeding of the International Conference on Software Engineering and Computer Systems (ICSECS), Kuantan, Malaysia, pp: 135-141, 2009.

[17] Bagui S., "Achievements and Weaknesses of Object-Oriented Databases". Journal of Object Technology, Vol. 2, No. 4, pp: 29-41, 2003. http://dx.doi.org/10.5381/jot.2003.2.4.c2

[18] Choi I., Bae S., Do N., and Yun M., "Backward Propagation of Engineering Constraints in Active Object-Oriented Databases". Proceedings of the 22nd International Conference on Computers and Industrial Engineering, Cairo, pp. 20-23, 1997.

[19] Jagannathan D., Ouck L., Fritchman L., Thompson P., and Tolbert D., "A Database System Based in the Semantic Data Model," Proceedings. ACM SIGMOD Conference, pp. 46-55, 1988.

[20] Choi M., Cho E., Park D., Bae J., Moon J, and Baik D., "A Synchronization Algorithm of Mobile Database for Ubiquitous Computing", 5th International Joint Conference on INC, IMS and IDC, pp.416-419, 2009.

[21] Abiona O., Oluwaranti A., Anjali T., Onime, C., Popoola E.; Aderounmu G., Oluwatope A., Kehinde L., "Architectural model for Wireless Peer-to-Peer (WP2P) file sharing for ubiquitous mobile devices", pp: 35-39, IEEE International Conference on Electro/Information Technology (EIT), 2009.

AUTHORS

Belal Zaqaibeh is with the Department of Computer Science, Faculty of Science and IT, Jadara University, Irbid, Jordan.

Firas Albalas is with the Computer Networks Department, Faculty of Science & IT, Jadara University, Jordan.

A. W. Awajan is with Al-Balqa' Applied University, Jordan.

Received 29 September 2011. Published as resubmitted by the authors 30 March 2012.