Enforcing User-Defined Constraints During the Run-Time in OODB

Belal Zaqaibeh[1], Hamidah Ibrahim[2], Ali Mamat[2], and Nasir bin Sulaiman[1]
[1]Faculty of Science and Information Technology, Zarqa Private University, Jordan
[2]Universiti Putra Malaysia, Malaysia

Abstract: *In this paper a run-time model is proposed. The run-time model enforces integrity constraints for attributes that are derived from composition and inheritance hierarchies. The run-time model is designed for enforcing the logical integrity constraints in object-oriented databases during the run-time. A new technique called detection method is designed to check the object meta data to detect and catch the object-oriented databases violation before it occurs. Furthermore, we have implemented the RTM and supported set of definitions that are for checking attribute values validity, object-oriented databases consistency, and also a method for verifying attribute values when inserting, deleting, and updating objects.*

Keywords: *Object-oriented databases, integrity constraints, constraints violation.*

Received December 12, 2006; accepted May 13, 2007

1. Introduction

The data accuracy, consistency, and integrity in Object-Oriented Databases (OODBs) are extremely important for developers and users. The current OODB Management Systems (OODBMSs) lack the capability of an ad-hoc declarative specification of enforcing and maintaining Integrity Constraints (ICs) that appear as a result of composition and inheritance hierarchies. ICs are conditions that data within a database must satisfy. Integrity maintenance or constraint enforcement is a set of activities to keep OODBs in a consistent state, where all instances in the OODB satisfy ICs.

This paper presents our contribution for this research, in which it clarifies the Assertion Model of ICs (AMIC). This paper is organized as follows. Section 2 presents the groundwork of our research. Section 3 explores the AMIC features, which includes Compile-Time Model (CTM) for Structural ICs (SIC), Run-Time Model (RTM) for Logical ICs (LIC), Object Meta Data (OMD), and Detection Method (DM). The IC enforcement for RTM is presented in section 4 when inserting, deleting, and updating objects. Section 5 presents the related and future work. Naturally, we end this paper by a conclusion in section 6.

2. Background

The increased emphasis on process integration is a driving force for the adoption of OODBSs [2]. For example, the CAD area is focusing heavily on using OODB technology as the process integration framework. Advanced office automation systems use OODBMSs to handle hypermedia data. Image processing and designing systems use OODB technologies for ease of use. All of these applications are characterized by having to manage complex, highly interrelated information, which is strength of OODBs. Clearly, relational databases technology has failed to handle the needs of complex information systems [7].

ICs in OODBSs 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 OODBMSs 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 naturally and maintained in OODBs, by capturing the violation using the type system and the class hierarchy [8]. Checking the ICs in OODBs is a fundamental dilemma in database design, because current OODBMSs lack the capability of an ad-hoc declarative specification of maintaining ICs that appear as a result of composition, association, and inheritance hierarchies. The constraints must be maintained in the backward direction along the class composition hierarchy as well as in the forward direction. The AMIC can represent ICs and their relationships over the composition, association, and inheritance hierarchies [10].

3. The ATM Components

The AMIC architecture consists of four components, as shown in Figure 1, namely: CTM, RTM, OMD, and DM. The IC enforcement and maintenance are divided into two phases (compile-time and run-time). In compile-time the CTM enforces and maintains SICs while the RTM enforces LICs during run-time phase. The CTM is performed only once for each OODB, while the RTM is performed whenever an update is submitted for processing.

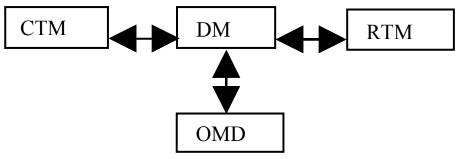


Figure 1. The AMIC architecture.

The separation of the system architecture into two phases reflects the fact that two different users may be involved. The user in the CTM, who interacts with the system and supplies it with added information, is referred to as the constraint designer. The user in the RTM, who uses the real system, is referred to as the end-user.

3.1. Complie-Time Model

The CTM is responsible for enforcing and maintaining ICs when a constraint designer submits a request during the compile-time. The CTM results a consistent constraints that are stored in the OMD. More details can be found in [9].

3.2. Detection Method

The DM is an overloaded method that can access and modify the OMD. The DM is designed for constraint validation checking purpose. Therefore, the DM has two functions that are differentiated from each other by their arguments. The first DM function is illustrated in Figure 2, which it has, the arguments: CID, AID, RCID, RAID, set of ACs, set of UDCs, and set of SCs.

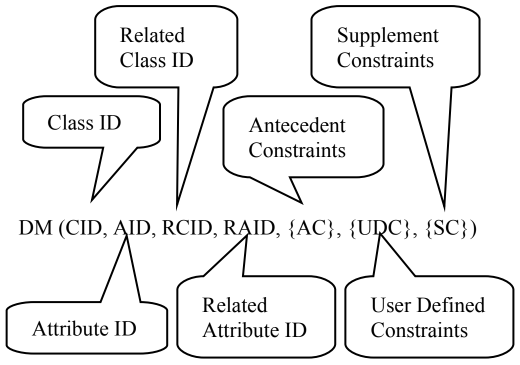


Figure 2. The DM heading.

The second DM method is illustrated in Figure 3 with the arguments: CID and AID.

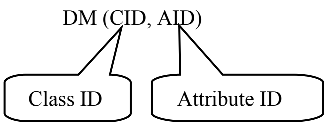


Figure 3. The heading of the overloaded DM.

The CID and AID are the composite key to reach the information about any attribute in the OMD. This information includes constraints base, derivation path, domains, derived attributes, and superclasses. The CID is a unique ID over the OODB level, this means the CID cannot be repeated even if a class is deleted then declared. The AID represents the ID for an attribute in a particular class, where the ID is unique under the class level; this means the AID can be repeated in 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.

3.3. The Object Meta Data

The OMD is the constraints map, it is responsible for building the specific knowledge base of the constraints of OODBs and is built once by the CTM [9]. 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, 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.

3.4. Run-Time Model

The RTM is responsible for enforcing the ICs, verifying transactions (inserting, updating, and deleting objects), checking constraint domains, and maintaining the unaccepted user request. The RTM communicates with the DM to get the constraints and attributes information. However, all transactions must remain the OODB in a consistent state. The RTM consists of the following components, as illustrated in Figure 4.

User Interface (UI) 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. Update Analyzer (UA) 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 (UC) with the involved attributes and constraints. The UC communicates with the DM to get the constraints and attributes path. Moreover, the UC checks the action knowledge and derives them to the Update Enforcer (UE). The UE is responsible for enforcing the ICs and verifying transactions by checking constraint domains.

The Update Maintenance (UM) is responsible for maintaining the unaccepted user request. The UM sends the actions that cannot be maintained to the Error Handler (EH) and the maintained action to the DM. The EH is responsible for handling errors then reports the violation knowledge. Each phase can encounter errors. However, after detecting an error, the present phase must somehow deal with that error.

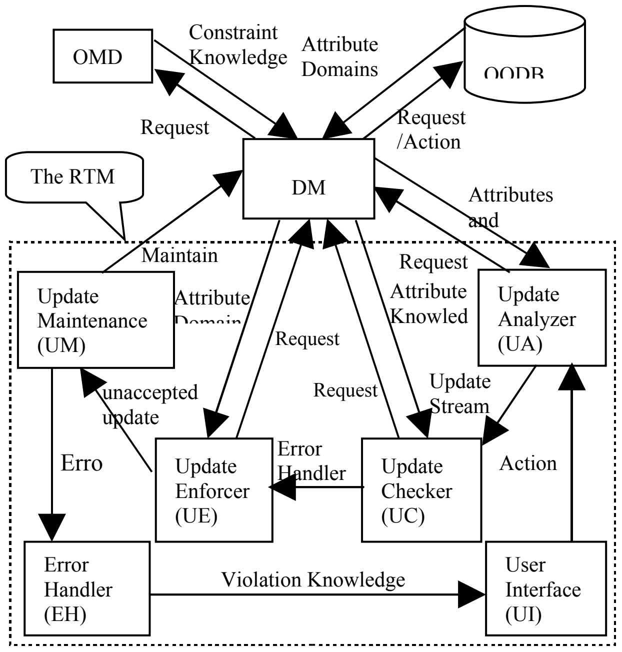


Figure 4. The RTM architecture in AMIC.

The DM has several functions that depend on the connection phase. Subsequently, the DM receives UA requests and accesses the OMD to get the attributes and constraint knowledge that is stored by the CTM during the compile-time [9] then sends them to the UA. Other functions are that receiving (UC and UE) requests and sending the attributes (knowledge and domains). If a user request does not violate the database and can be enforced in the UE then the DM allows the user request to be performed. Moreover, the DM receives the actions from the UM if a user request needs to be maintained whereas the UM does the maintenance.

Typically, algorithm 1 presents the main functions of the RTM for enforcing ICs. The CTM and RTM are integrated and implemented in the AMIC. The RTM receives *u_r* through the UI, and then analyzes it in the UA to determine the action type (insert, update, or delete) with the help of the DM, whereas the UA communicates with DM by sending requests and receiving knowledge about the involved attributes and constraints. Subsequently, The UA sends requests of *u_r* to the DM to get the involved attributes, constraints, and their paths. Moreover, after all required *u_r* information is collected, the UA sends streams of *u_r* to UC for checking purpose.

```
A. Input: User request (u_r).
B. Output:
     1. The results of performing the u_r,
     2. Violation knowledge if the u_r is invalid.
C. Steps:
     Start
     Repeat
     UI ← u_r,
     UA ← UI(u_r)
     UA ← OMD ↔ DM(CID, AID, RCID, RAID,
        {AC}, {UDC}, {SC})
     UC ← UA
     UE ← DM ↔ UC
     For i =1 to n Do
          If Verify(Val, O.A_i) Then
          DM ← UE
            Else
          UM ← UE
          If UM (u_r) Then
                 goto Step 5
                 Else
                 EH ← UM
                 goto Step 21
               End If
          End If
     Until u_r = ∅
     End
```

Figure 5. Algorithm 1 RTM.

The UC communicates with the DM to get attribute domain then sends the *u_r* with the related (antecedent/ supplement) attributes and constraints to the UE. Subsequently, the *u_r* is enforced in the UE by verifying the new values with the attribute domain. However, if the *u_r* is verified and remained true then the UE will send the *u_r* to the DM otherwise to the UM for maintenance purpose. If the *u_r* can be maintained then the UM will send it to the DM otherwise to the EH to show the violation knowledge and abort the *u_r*.

4. Integrity Enforcement in RTM

In this section, we discuss integrity enforcement and maintenance when objects are inserted, updated, or deleted. Enforcing ICs in RTM occurs during the run-time, so the maintenance of RTM is required whenever event is submitted. Therefore, there are two general steps to be performed. First, all constraints that would be violated must be found. Second, the AMIC should determine what action must be taken.

- *Definition 1*: values validity, Let *O* be an object, *C* a constraint on *O*, *D* the domain of *C*, *A* an attribute in *O* where $Ai \in A$, Val is a value of *Ai*, and *Verify(Val*, *O.Ai)* returns true if *Val* is a value that accepted in *Ai* and δ *(C)* is true otherwise false. Before inserting a new object, all the values in each attribute must be verified to satisfy its constraints. Therefore, if an invalid value is assigned to an attribute then the AMIC will reject it.
- *Definition 2*: database consistency, Let *D* be an OODB that has an object *O*, a constraint *C* on *O*, and attribute *A*. Also Let *Val* be a value to be

inserted into *A* and *D*+ or $\tilde{D}$ is the new *D* state after inserting, deleting, or updating *O*. Subsequently, if *Verify(Val*, *O.A)* remains true then *D* is consistent and is denoted by *D*+ otherwise *D* is inconsistent and denoted by $\tilde{D}$. Therefore,

$$IF \cap_{i=1}^{n} \; Verify(Val,0.A)\, is\, True\, Then\, D^{+}\, e \tag{1}$$

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

- *Definition 3*: calling function, Let *D* be an OODB and *Calli* ∈ *Call* be a function *call* for the DM(CID, AID), *i*=1, 2, …, *n*, and the OODB is *D*+ if *Calli* remains true otherwise $\tilde{D}$. Therefore,

$$If \cap_{i=1}^{n} Call_{1}\; is\, True\, Then\, D^{+}\, elas\, D^{-} \tag{2}$$

Figure 6 shows a composition, association, and inheritance hierarchies among *Person*, *Child*, and *Meal* classes. *Child* has composed and inherited *Person* and also is associated with *Meal*.

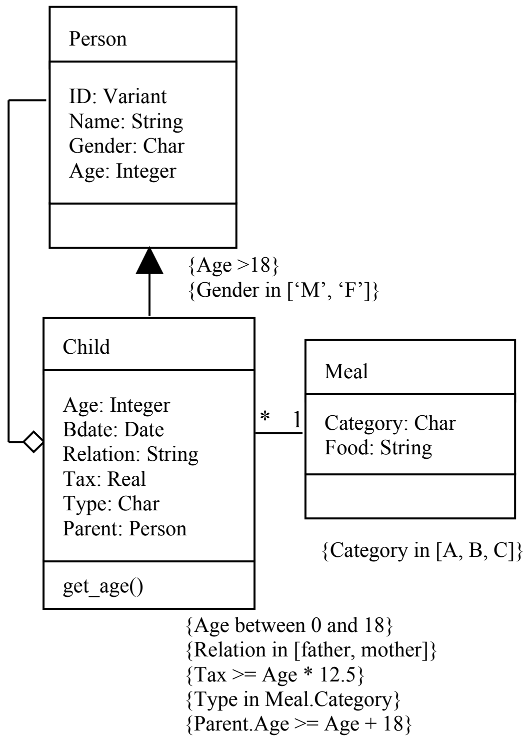


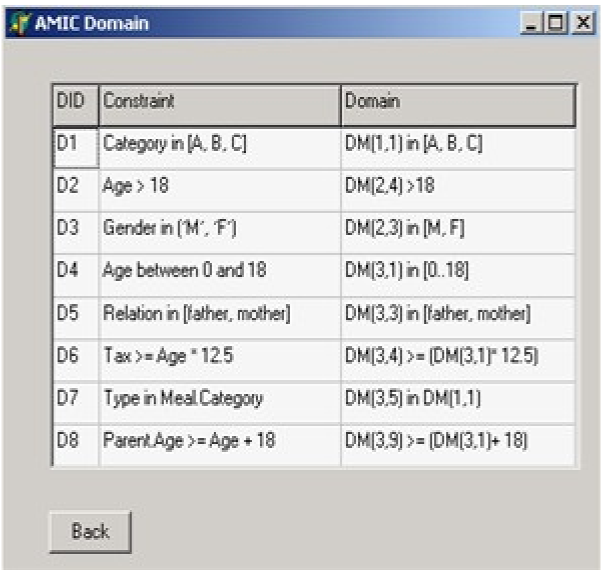
Figure 6. Classes and their constraints.

As mentioned earlier, the AMIC generates the OMD in the CTM. Therefore, the AMIC will call the DM 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:

$$DM(CID,\ AID,\ RCID,\ RAID,\ AC,\ UDC,\ SC) \tag{3}$$

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 are dealing with UDCs, and regardless whether the classes are designed in a good or bad design, all constraints and domains are verified, optimized, and collected in the OMD, as shown in Figure 7.



DID	Constraint	Domain
D1	Category in [A, B, C]	DM[1,1] in [A, B, C]
D2	Age > 18	DM[2,4] >18
D3	Gender in ('M', 'F')	DM[2,3] in [M, F]
D4	Age between 0 and 18	DM[3,1] in [0..18]
D5	Relation in [father, mother]	DM[3,3] in [father, mother]
D6	Tax >= Age * 12.5	DM[3,4] >= (DM[3,1]* 12.5)
D7	Type in Meal.Category	DM[3,5] in DM[1,1]
D8	Parent.Age >= Age + 18	DM[3,9] >= (DM[3,1]+ 18)

Figure 7. The optimized domains in the OMD.

Also, the attributes and their relationships are stored in the OMD, as shown in Figure 8. The association between *Meal* and *Child* does not inherit attributes from class to another. In the contrary of that the inheritance and composition between *Person* and *Child* propagate constraints and attributes to *Child*, as the RCID and RAID illustrate for *Child*.

All processes to create the OMD are categorized under the CTM. In the RTM, the AMIC enforces the data integrity whenever a request for inserting, deleting, or updating objects occurs.

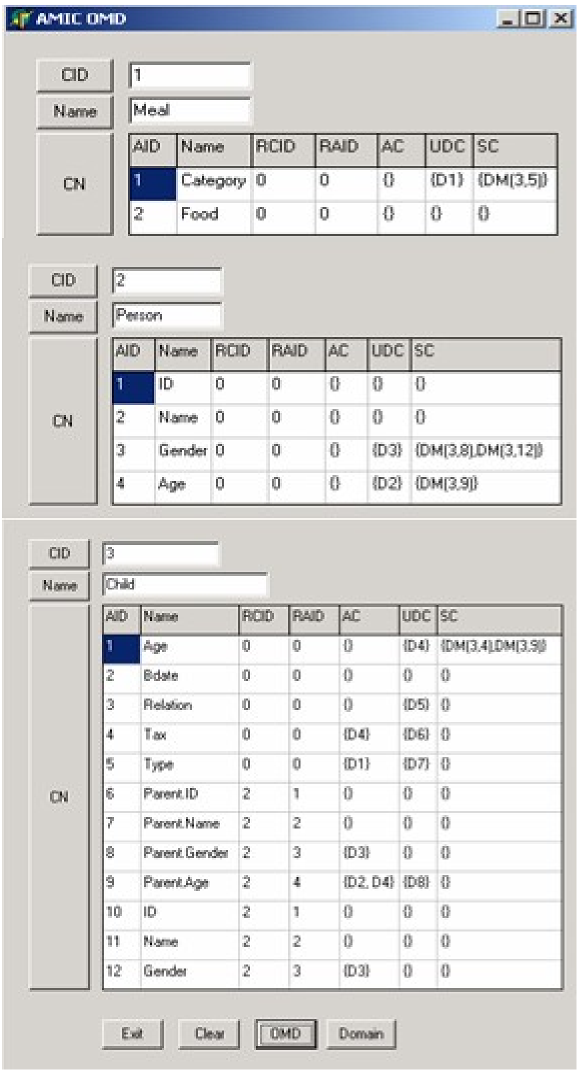
4.1. Inserting Object

When inserting a new object, all constraints in OMD that are related to that object must be checked to verify the new data state. By referring to Figure 6, we can declare three different objects (OM, OP, and OC) from the classes (*Meal*, *Person*, and *Child* respectively). Each object carries a class members and a reference to the OMD. A reference supervises the connection between the object and its related constraints in the OMD. Let us consider the following cases of insertion:

A. Case 1: Inserting a new data into OM

- OM(Category, Food)
- This requires to enforce the DM(1,1) and DM(1,2). Subsequently, the AMIC will call the DM as follows:
  - *Call1*: DM(1, 1, 0, 0, {}, {D1}, {DM(3,5)})
  - *Call2*: DM(1, 2, 0, 0, {}, {}, {})

The DM(1,1) must be in *D1* domain to satisfy its constraint as shown in *Call1*, otherwise the insertion request will be rejected and the violation path will be showed. Moreover, the DM is not called for the SC of DM(3,5) because the Meal is not propagated from any class. For the DM(1,2), there is no AC, UDC, or SC to be checked as shown in *Call2*, so the new data will be accepted if it satisfies *Verify(Val, DM(1,1))* and *Verify(Val, DM(1,2))*.



CID	1
Name	Meal

CN:

AID	Name	RCID	RAID	AC	UDC	SC
1	Category	0	0	{}	{D1}	{DM(3,5)}
2	Food	0	0	{}	{}	{}

CID	2
Name	Person

CN:

AID	Name	RCID	RAID	AC	UDC	SC
1	ID	0	0	{}	{}	{}
2	Name	0	0	{}	{}	{}
3	Gender	0	0	{}	{D3}	{DM(3,8),DM(3,12)}
4	Age	0	0	{}	{D2}	{DM(3,9)}

CID	3
Name	Child

CN:

AID	Name	RCID	RAID	AC	UDC	SC
1	Age	0	0	{}	{D4}	{DM(3,4),DM(3,9)}
2	Bdate	0	0	{}	{}	{}
3	Relation	0	0	{}	{D5}	{}
4	Tax	0	0	{D4}	{D6}	{}
5	Type	0	0	{D1}	{D7}	{}
6	Parent.ID	2	1	{}	{}	{}
7	Parent.Name	2	2	{}	{}	{}
8	Parent.Gender	2	3	{D3}	{}	{}
9	Parent.Age	2	4	{D2, D4}	{D8}	{}
10	ID	2	1	{}	{}	{}
11	Name	2	2	{}	{}	{}
12	Gender	2	3	{D3}	{}	{}

Figure 8. The OMD for child database.

B. Case 2: Inserting a new data into OP

- OP(ID, Name, Gender, Age)
- This requires to enforce the DM(2,1), DM(2,2), DM(2,3), and DM(2,4). Subsequently, the AMIC will call the DM as follows:
  - *Call3*: DM(2, 1, 0, 0, {}, {}, {})
  - *Call4*: DM(2, 2, 0, 0, {}, {}, {})
  - *Call5*: DM(2, 3, 0, 0, {}, {D3}, {DM(3,8), DM(3,12)})
  - *Call6*: DM(2, 4, 0, 0, {}, {D2}, {DM(3,9)})

In *Call3* and *Call4* there are no UDCs that were defined so the AMIC accepts data for these attributes. In *Call5* there is a UDC which is *D3* and also SCs. Thus, the new data in OP for this attribute must satisfy *D3* else it will not be accepted. Moreover, in *Call6* there is a UDC D2 that enforces DM(2,4) data. Typically, the AMIC will not check the SCs for the DM(3,8), DM(3,12), and DM(3,9) because *Person* is not propagated from any class.

C. Case 3: Inserting a new data into OC

- OC(Age, Bdate, Relation, Tax, Type, Parent.ID, Parent.Name, Parent.Gender, Parent.Age, ID, Name, Gender)
- This requires to enforce the DM(3,1), DM(3,2), DM(3,3), DM(3,4), DM(3,5), DM(3,6), DM(3,7), DM(3,8), DM(3,9), DM(3,10), DM(3,11), and DM(3,12). Then, the AMIC will call the DM as follows:
  - *Call7*: DM(3, 1, 0, 0, {}, {D4}, {DM(3,4), DM(3,9)})
  - *Call8*: DM(3, 2, 0, 0, {}, {}, {})
  - *Call9*: DM(3, 3, 0, 0, {}, {D5}, {})
  - *Call10*: DM(3, 4, 0, 0, {D4}, {D6}, {})
  - *Call11*: DM(3, 5, 0, 0, {D1}, {D7}, {})
  - *Call12*: DM(3, 6, 2, 1, {}, {}, {})
  - *Call13*: DM(3, 7, 2, 2, {}, {}, {})
  - *Call14*: DM(3, 8, 2, 3, {D3}, {}, {})
  - *Call15*: DM(3, 9, 2, 4, {D2, D4}, {D8}, {})
  - *Call16*: DM(3, 10, 2, 1, {}, {}, {})
  - *Call17*: DM(3, 11, 2, 2, {}, {}, {})
  - *Call18*: DM(3, 12, 2, 3, {D3}, {}, {})

The analysis is clarified in the following points:

- With no constraints: In *Call8*, *Call12*, *Call13*, *Call16*, and *Call17* there are no ACs, no SCs, and no UDCs. We notice here from the RCID and RAID that the *Call8* is verifying the attribute DM(3,2) which exists in the current class. On the other hand, the *Call12* and *Call13* are verifying the attributes that are inherited from composition hierarchy. And also *Call16* and *Call17* are inherited from inheritance hierarchy.

- UDCs: In *Call9* there is a UDC which is *D5*, so the AMIC will call and verify the DM(3,3) as it must be "father" or "mother" to be accepted.
- ACs: In *Call14*, and *Call18* there is an antecedent *D3* that is derived from the composed class for *Call14* and from the inherited class for *Call18*. Subsequently, because *D3* is the domain of DM(2,3) so this requires calling the DM for *Call5* as the DM(3,8) and DM(3,12) must be "*M*" or "*F*" to be accepted.
- ACs and UDCs: In *Call10*, *Call11*, and *Call15* there are ACs and UDCs. We notice here *Call10* requires verifying the *D4* and *D6* that are declared in the current class. And also *Call11* requires verifying the domain *D1* that is derived from the associated class. Furthermore, *Call15* requires verifying the *D2* that is derived from the composed class. However, for *Call10* the AMIC will verify the DM in *Call7*, for *Call11* the AMIC will verify the DM in *Call1*, and for *Call15* the AMIC will verify the DM in *Call6*, and *Call7*.
- SCs and UDCs: In *Call7* there is a UDC which is *D4*. AMIC will verify the attribute DM(3,1) whether it satisfies *D4* or not. Consequently, the AMIC will verify the SCs DM(3,4) and DM(3,9) by *Call10* and *Call15*.

4.2. 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. For instance:

- Deleting the OM required to verify the supplement DM(3,5) and this leads to verify *Call11*,
- Deleting the OP required to verify the supplements DM(3,8), DM(3,9) and DM(3,12) and this leads to verify *Call14*, *Call15* and *Call18*,
- Deleting the OC required to verify the supplements DM(3,4) and DM(3,9) and this leads to verify *Call10* and *Call15*.

4.3. Updating Object

Updating objects requires keeping the current database state *D* until verifying the ICs in *D+*. Based on Figure 5, let us consider the following cases that may occur during the RTM and may affect *D*.

A. Case 1, Update statement that modifies *OC.Name* and *OC.Parent.Name* with the values *Pname* and *Cname* respectively (*Pname* and *Cname* are two variables with new values). Then, The attribute *OC.Parent.Name* DM(3, 7, 2, 2, {}, {}, {}) is derived from composition hierarchy and *OC.Name* DM(3, 11, 2, 2, {}, {}, {}) is derived from inheritance hierarchy, and these are *Call13* and *Call17* respectively. Each DM has no UDCs, so it is impossible to have *D-* if and only if *Verify(Pname, DM(3,7))* and *Verify(Cname*, *DM(3,11))* remain true.

B. Case 2, Update statement that modifies *OC.Relation* value with *Newrelation* (Newrelation is a variable with a new value). Then, The *OC.Relation* DM(3,3) must satisfy *D5* to enforce ICs as shown in *Call9*. If *Verify(Newrelation, DM(3,3))* remains true then the AMIC results *D+* else aborts the user request and keeps the violation path.

C. Case 3, Update statement that modifies *OC.Parent.Age* with *Newage* value (Newage is a variable with a new value). Then, The AMIC gets DM which is DM(3,9) for *OC.Parent.Age* attribute. To enforce the ICs of DM(3,9) the AMIC verifies *Call15* which it requires to satisfy *D2*, *D4* and *D8* to remain *D+*. Since *D2* is the domain of the constraint that has been declared on DM(2,4) then AMIC will verify *Call6*. So if *Call6* remains true the AMIC will verify the next domain otherwise abort the current user request. For *D4* the AMIC calls DM(3,1) and verifies *Call7*. Moreover, the AMIC verifies *Call10* because DM(3,1) has SCs. If each of *Call15*, *Call6*, *Call7* and *Call10* remains true as follows:

- *D* is $\bigwedge_{i=1}^{n} Call i$
- *D* is $Call15 \wedge Call6 \wedge Call7 \wedge Call10$
- *D* is true
- *D* is *D+*

This produces a consistent database. By referring to definition 3 the AMIC results D+ and accepts the updating statements. But if any of the Call15, Call6, Call7, or Call10 returns false then:

- *D* is $\bigwedge_{i=1}^{n} Call i$
- *D* is $Call15 \wedge Call6 \wedge Call7 \wedge Call10$
- *D* is false
- *D* is *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 AMIC, as the current OODBMS and object-oriented applications do not have the ability to support the violation path.

5. Related Work

The proper handling of ICs is essential to any data storage and management. Handling ICs is an essential premise to managing semantically rich data [7]. In OODBs, checking the ICs is a fundamental problem in the database design [7]. The automated verification of constraints and their enforcement provided by current OODBMSs is limited [7, 6] due to the user participation is required.

OODBMSs do not have adequate support for certain types of constraints especially the ones defined in a class composition and inherence hierarchies [1, 7, 5, 3]. 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. OODBs.

More work can be done when copying an object of a superclass to another object of a subclass and vise versa. For such problem downcasting 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.

6. Conclusion

This paper has shown the RTM properties, specifications, and architecture. The AMIC has made a big challenge in the OODM environment as it can represent constraints and complex relationships among attributes and classes that are derived from composition and inheritance hierarchies, whereas the current OODBs are deficient in such properties.

A set of definitions is supported for checking attribute values validity, OODB consistency, and also a method for verifying attribute values when inserting, deleting, and updating objects. Thus, the AMIC is able for enforcing and maintaining ICs in SICs by the CTM and LICs by RTM. The CTM is designed for enforcing the constraint base during the compile-time while the RTM is designed for enforcing the data integrity during the run-time.

The OMD keeps track the constraint paths in the backward direction as well in the forward directions, keeps the constraint knowledge to ease accessing them, and includes knowledge about attributes and their relationships, constraints, and domains. Typically, RTM is an automated model that can enforce ICs the run-time.

References

[1] Bagui S., "Achievements and Weaknesses of Object-Oriented Databases," *Journal of Object Technology*, vol. 2, no. 4, pp. 29-41, 2003.

[2] Brown P., *Object-Relational Database Development*, Addison-Wesley, 2001.

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

[4] David W., *Object Database Development Concepts and Principles*, Addison-Wesley, 1998.

[5] Do N., Choi I., 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.

[6] Eick C. and Werstein P., "Rule-Based Consistency Enforcement for Knowledge-Based Systems," *The IEEE Transactions of Knowledge and Data Engineering*, vol. 5, no. 1, pp. 52-64, 1993.

[7] 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.

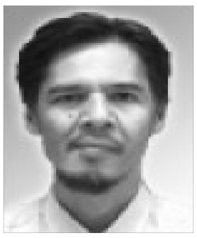
[8] Urban S. and Wang A., "The Design of a Constraint/ Rule Language for an Object-Oriented Data Model," *Elsevier Science System Software*, vol. 28, no. 3, pp. 203-224, 1995.

[9] Zaqaibeh B., Ibrahim H., Mamat A., and Sulaiman M., "An Assertion Model for Controlling Integrity Constraints in an OODB," *in Proceedings of the International Conference on Informatics and RWICT*, pp. 413-421, 2004.

[10] Zaqaibeh B., Ibrahim H., Mamat A., and Sulaiman M., "Enforcing and Maintaining Constraints Base During the Compile-Time," *Journal of WSEAS Transactions on Computers*, 99-357, 2006.

Belal Zaqaibeh received his BSc degree with the first honor degree in computer science from Irbid National University, Jordan, in 1998. In 1999, he was the manager of Makkah Center for Computer. In 2000, he continued to graduate school at Universiti Putra Malaysia (UPM) and received his MSc in distributed computing in 2001 and his PhD in object-oriented databases in 2006. In 2006, he joined Zarqa Private University, Jordan, where he is currently working as an assistant professor of computer science. His research interests include object-oriented databases, mobile databases, integrity constraints, and object-oriented software engineering.

Hamidah Ibrahim is currently an associate professor at the Faculty of Computer Science and Information Technology, Universiti Putra Malaysia. She obtained her PhD in computer science from the University of Wales Cardiff, UK in 1998. Her current research interests include databases, transaction processing, and knowledge-based systems.



Ali Mamat is an associate professor at Computer Science Department, Universiti Putra Malaysia. He obtained a PhD in computer science from University of Bradford, UK in 1992. His research interests include databases, XML, and semantic web.

Nasir bin Sulaiman is a lecturer in Computer Science in Faculty of Computer Science and Information Technology, UPM. He has been appointed as an associate professor in 2002. He obtained PhD in neural networks simulation from Loughborough University, UK in 1994. His research interest includes neural networks theory and applications, intelligent software agents, and data mining.