I.J. Information Technology and Computer Science, **2013, 05, 74-84**
Published Online April 2013 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2013.05.10

Semantic Multi-granular Lock Model for Object Oriented Distributed Systems

V.Geetha
Department of Information Technology, Pondicherry Engineering College, Puducherry, India
E-mail: vgeetha@pec.edu

N.Sreenath
Department of Computer Science & Engineering, Pondicherry Engineering College, Puducherry, India
E-mail: nsreenath@pec.edu

***Abstract*—** In object oriented distributed systems (OODS), the objects are viewed as resources. Concurrency control techniques are usually applied on the database tier. This has the limitations of lack of support of legacy files and requirement of separate concurrency control mechanisms for each database model. Hence concurrency control on the objects at server tier is explored. To implement concurrency control on the objects participating in a system, the impact of method types, properties and class relationships namely inheritance, association and aggregation are to be analyzed. In this paper, the types and properties of classes and attributes are analysed. The semantics of the class relationships are analysed to ascertain their lock modes, granule sizes for defining concurrency control in OODS. It is also intended to propose compatibility matrix among all these object relationships.

***Index Terms*—** Object Oriented Distributed Systems, Concurrency Control, Multi Granular Lock Model, Class Relationships, Design Time Transactions, Run Time Transactions

I. Introduction

Object Oriented Database Management Systems (OODBMS) provides better complex data modeling support for the newly emerging distributed applications than relational databases. OODBMS is used as persistent data store for distributed systems. It resides in the database tier. However in distributed systems, the server tier is implemented as procedures. This requires a conversion between procedural paradigm to object oriented paradigm and vice versa for all the communications between server tier and data base tier. Further each of the database models in distributed systems has its own concurrency control mechanisms that cannot be adapted to any of the other models. The concurrency control policies are defined only for the database tier. The server tier has no control over the concurrency control. This introduces a restriction of using only the refined persistent data store for the domain data like database management systems. Primitive data stores like files are not supported in distributed systems and hence they cannot be reused.

The above limitations can be overcome in OODS. In OODS, the server tier is also implemented as objects. So the conversion of data format between server tier and data base tier is not necessary, if OODBMS provides persistent data storage. However conversion of data format is required, if other database models are used. Hence the possibility of providing a common concurrency control mechanism that is independent of the persistent data store type is explored in [1] by shifting the concurrency control from database tier to server tier.

In OODS, objects are the reusable data sources. The clients can access the data from the data store in database tier only through the objects in the server tier. Hence a common concurrency control mechanism can be defined for the objects in the server tier. The other advantage of this shift in the concurrency control to the server tier allows usage of legacy data stores.

Already semantic concurrency control mechanisms have been proposed for object oriented data bases by exploiting the object oriented paradigm features. They out-perform the conventional concurrency control mechanisms for OODBMS. Hence the feasibility of extending the same mechanisms to OODS may be analyzed.

OODS support continuously evolving domains in which the services are frequently enhanced to provide better client support. Hence transactions providing run time services (run time) and design updates (design time) are to be supported.

Though concurrency control mechanisms in OODBMS can be considered, they cannot be extended as they are in OODS. This is because query languages are used to access databases. But in OODS, object oriented programming languages like C++, Java are used to make client requests. Then lock types and granularities of resources are to be ascertained from the client code. The doc tools like docC++, Javadoc can be used for identifying the method type and properties. Following this, the compatibility matrix used in OODBMS can be considered for adoption in OODS.

In OODBMS, lock modes are defined only for concrete classes. The lock modes for abstract classes are not ascertained. The compatibility matrices for inheritance and aggregation have been defined for OODBMS. To use those matrices, lock types and granule sizes are to be determined for inheritance and aggregation (composition) using the classification of class method types and properties proposed in [2, 3]. Association is one the important relationships frequently used to relate objects participating in a domain. In [4], they have proposed directed graph based association algebra for query processing and optimization of objects in object oriented databases. But so far, the types, properties and attributes of association have not been explored for their probable impact on concurrency or concurrency control. The lock modes, granule sizes and lock compatibility for association have not been explored so far.

In the following section, a semantic concurrency control technique is proposed for object oriented distributed systems. It is done in two steps namely (1) defining lock types and granularity for all types of classes and their relationships (2) extending the compatibility matrix defined for OODBMS to OODS. Section 3 concludes the chapter.

II. Defining Lock Types and Granularity for Classes and Their Relationships

2.1 Object Oriented Concepts

This section revisits the object oriented concepts related to the research work. The types and properties of object methods are explained first. Then the semantics of class types, attribute types and class relationships with respect to locking is discussed.

The client requests are satisfied by executing the methods defined in the object. These methods need to operate on the data to satisfy the request. The methods not only have types but also properties. Depending on the type of methods, the read or write operations can be ascertained. Then concurrency control mechanisms can be defined whenever there are R-W and W-W conflicts. In [2] the object methods are classified into three types:

1. **Query method:** returns some information about the object being queried. It does not change the object's state. There are four main query method types:- *Get method, Boolean query method, Comparison method and Conversion method.*
2. **Mutation method:** changes the object's state (mutates it). Typically, it does not return a value to the client. There are three main mutation method types:- *Set method, Initialization method and Command method.*
3. **Helper methods:** performs some support task for the calling object. There are two types of helper methods: - *Factory method and Assertion method.*

Apart from types, a method also has properties [3]. Example of method properties are whether the method is *primitive or composed*, whether it is available for *overriding through subclasses (hook method)*, or whether it is a mere *wrapper around a more complicated method (Template method)*. A method has exactly one method type, but it can have several properties. Method types and properties are orthogonal and can be composed arbitrarily.

Two types of classes are defined in object oriented systems namely *Abstract* and *Concrete classes*. Abstract classes are usually used to define the class template. Instances are not created from this type of classes. Usually they act as base classes from which one or more concrete classes are derived. Concrete classes are classes defined mainly to create instances. They support all types of methods to create, query, mutate and delete objects. The locks on concrete classes depend on the type of member method which is invoked. Both read (S) and write (X) locks must be available for them at both design time as well as runtime. So lock types for both abstract and concrete classes are to be ascertained.

In OODBMS, only instance level attributes are referred. The scope of values of these attributes is restricted to the state of the object in which they are present. They are mutually independent and directly inaccessible by other objects of the same class. In OODS, instance level attributes as well as class level attributes are present. The class level attributes are shared by all instances of a class. They are also called as static attributes of a class. For e.g., *nextregno* can be defined as a static member in the student class to generate the next register number for a new student object. Hence the smallest granule size for instance level attributes could be object or individual attributes, whereas the granule size of class level attribute can be as small as a class.

As mentioned earlier, the classes are related by inheritance, aggregation and association relationships. The inheritance relationship also called as "IS A" relationship is sub divided into single inheritance, multi level inheritance, multiple inheritance, hierarchical inheritance and hybrid inheritance. The inheritance relationship except multiple inheritance can be represented using tree structure and is called class hierarchy. The inclusion of multiple inheritance will lead to network structure and is called class lattice.

The aggregation also called as "HAS A" relationship defines the containment of component objects in a composite object. The composite object uses the services of component objects to provide its service.

There are two types of aggregation namely strong and weak aggregation. The weak aggregation is a subtype of association and hence the rules used for association can also be extended to this. The strong aggregation is also called as composition and defines “PART OF” relationship. The composition [5] can be classified into dependent or independent based on the dependence of creation and deletion of component objects on composite objects. The composition is also classified into shared or exclusive based on the possibility of sharing component objects by more than one composite object.

The association relationship defines the USING relationship, where one or more objects use the service of an object. Since it is an object relationship, a binary association can be treated as shared composition with single component and N-ary association can be treated as shared composition with multiple component objects. The rules defined for composition may be extended to association.

In [5] and [6], they have explored the types and properties of inheritance and aggregation. However it is worth noting certain points regarding these relationships:

1. Transactions can request a single object or all the objects of a class based on the member function present in it. The property of the member function may be instance level or class level [3]. In [6], it states that when class level methods are called, instead of setting individual locks on all objects, a single lock on its class may be set to minimize the lock escalation.

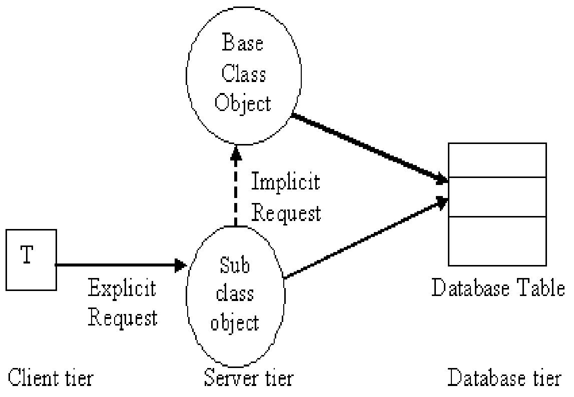


Fig. 1: Locking the sub class object with its base class object to maintain consistency

2. When a transaction requests a sub class object (fig 1), the sub class object and its corresponding base class object mapping to the same record in a database table must also be locked to maintain consistency. Hence base class object is an implicit resource needed for a transaction, when a transaction makes explicit resource request to sub class object. However when base class objects are requested, sub class objects need not be locked.

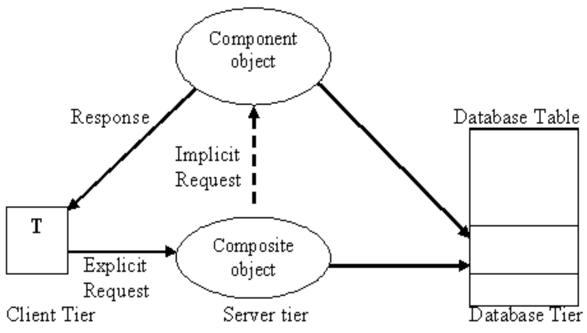


Fig. 2: Locking the composite object with its component object to maintain consistency

3. When a transaction requests a composite object, its component objects also need to be locked. In aggregation, component objects constitute composite object. Hence component objects are implicit resources to composite object (explicit resource). The composite object gets the request and forwards it to component object, if the service is implemented in component object. The component object provides the service to the transaction as in figure 2.

4. In association, when a transaction calls an associative object, it may access associated object to provide the service. Then associated object needs to be locked along with the requested object to maintain consistency as in figure 3.

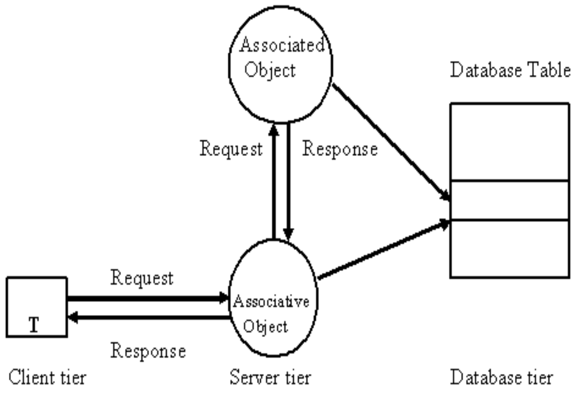


Fig. 3: Locking the associative object with its associated object to maintain consistency

Association differs from Inheritance and Aggregation relationships in the following ways:

- Association requires several qualifying attributes to completely define itself, unlike “IS-A” and “HAS-A” relationships that are complete and semantically strong.
- In Inheritance and Aggregation, the cardinality of the relationship is usually 1.But in association; the cardinality can range from 0 to many. Hence a policy must be decided to fix the granule size.
- Reflexive association is present only in association, in which one object may associate with 0 or more objects of the same class. This leads to self looping.

- Usually inheritance and aggregation are static. These relationships are decided at design time. But association can be static or dynamic.

Association is classified in to the following categories[7,8]:

1. Direct vs. Indirect Association:

In direct association, the two classes are directly linked. This will be usually binary association.

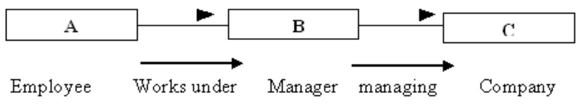


Fig. 4: example for indirect association

In fig 4, the association between A, B and B, C are direct. But the association between A and C is indirect. This implies that if class A is requested, then B is also to be requested. This is because B is directly associated with A and A might need the services of B. But B is associated with C. This implies that B might use the services of C to serve A. Hence A is indirectly associated with C. When B is locked along with A, C also needs to be locked. This association type decides the extent of locking.

2. Binary Vs N-ary Association:

Binary association is association between two classes. If more than two classes are associated, then it is called N-ary association. N-ary association is difficult to implement as it is. Hence it is implemented as a collection of binary associations. In fig 5a

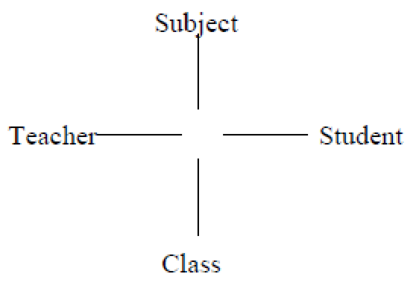


Fig. 5a: N-ary Association

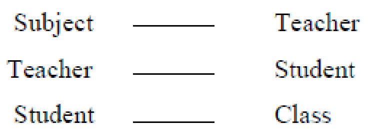


Fig. 5b: equivalent binary association

3. Referential Vs Dependent Association:

In referential or independent association, the association is logical. The associated classes are called as target and source classes. Target class is connected to source class which provides service. This typically defines "USING" relationship. When source classes are removed, the target classes are not removed. They are independent of each other.

Alternately, dependent association is physical. Here the classes are called producer and client. If producer is removed, the client also ceases to exist. In other words, client depends on server for its existence. This imposes constraints on creation and deletion of client on producer.

4. Shared vs. Exclusive Association:

In this type, the association is either dedicated to one class or shared with many classes.

5. Static vs. Dynamic Association:

In [9], it is stated that association can have static or permanent links (long term association) or dynamic links (short term association). Static links are defined at design time. But Dynamic links are transient, contextual and initiated only on request. Hence request for dynamically associated classes are deferred till runtime.

6. Reflexive Association:

This is a rarity in association itself. An object can be a client of other objects in the class. In fig 6,

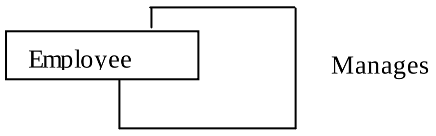


Fig. 6: example for reflexive association

A supervisor, who is also an instance of employee, manages other employees. This is called as self looping.

7. Inherited Association:

In fig 7, the association between subject and student is inherited to the derived class PG Student also. This lets redefinition of the association between student and subject.

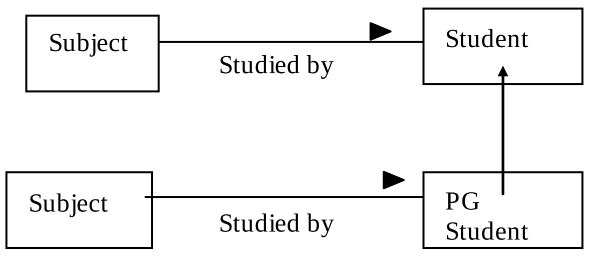


Fig. 7: example for inherited association

Any association is expected to define the following attributes to be semantically complete [10].

1. Role name: Two classes may have more than one association. This helps to select a specific association at a time between the two associated classes. This helps to deduce what attributes are going to be accessed for a particular association. Then concurrency may be increased.

2. Interface specifier: Along with role name, this also helps to identify attributes required, the services (methods) provided in a specific association.

3. Visibility: Specifies the access rights to other attributes and methods in the class. A transaction in OODS is typically constituted of interfaces. An interface may contain one or more methods or member functions of the implementing class. Then it is required that these methods are declared as 'public'. Otherwise they are hidden from the client and their request will not be satisfied.

4. Cardinality/ Multiplicity: Cardinality specifies the correspondence between the associated classes. This can be used to deduce granule size.

The above mentioned factors can be utilized while defining lock model for objects related by association. So far, the association relationship is not considered because of its inability to completely define the relationship semantically.

2.2 Defining Lock Types and Granularity for Attributes and Classes

In OODBMS, only instance level attributes are referred. The scope of the values of these attributes is restricted to the state of the object in which they are present. They are mutually independent and directly inaccessible by other objects of the same class. In OODS, instance level attributes as well as class level attributes are present. The class level attributes are shared by all the instances of a class. They are also called as static attributes of a class. For e.g., *nextregno* can be defined as a static member in 'student' class to generate the next register number for new student object. Hence the smallest granule size for instance level attributes could be object or individual attributes, whereas the granule size of class level attribute can be as small as only a class. Table 2a gives their granularity.

Abstract classes are usually used to define the class template. Instances are not created from this type of classes. Usually they act as base classes from which one or more concrete classes are derived. So at runtime, they should be locked only in S (read) mode. This is because the concrete classes that are inherited from this abstract class would be reading them. As they do not create instances (objects) and thereby do not affect the state of the system. However at design time, modifications may be done to the abstract class by inserting new methods or attributes, modifying the signature of the existing methods or modifying the data types of the attributes or deleting one or more attributes and/or methods. Hence the design time clients must be allowed to lock the abstract class by both S (read) and X (write) locks. It is also worth noting that the smallest accessible granule of abstract class is a class.

Table 1: Lock types for types of classes

Class type	Lock type (Design time)	Lock type (Runtime)
Abstract class	S/X	S
Concrete class	S/X	S/X

Concrete classes are classes defined mainly to create instances. They have all types of methods to create, query, mutate and delete objects. So the locks on concrete classes depend on the type of method which is invoked. So both S and X locks must be available for them at both design time as well as runtime. [5, 6, 11,12,13] address only concrete classes. Their granularity can be as small as attribute [12]. Table 1 summarizes the lock types allowed for the types of classes at design time and run time and table 2b summarizes their granularity.

Table 2a: Granularity of attributes

Type of class	Granularity
Abstract class	Class hierarchy/Class
Concrete class	Class/ Instance/Attribute

Table 2b: Granularity of classes

Type of Attributes	Granularity
Instance level	Instance/Attribute
Class level	Class

2.3 Types and Granularity of Locks Based on Method Types for Inheritance

Based on the definitions of the method types and its properties in section 2.2, the locks can be determined for inheritance relationship as given below. In [14], they have defined the following types of lock modes for coarse and fine granules for relational databases. It is extended to object oriented databases as follows.

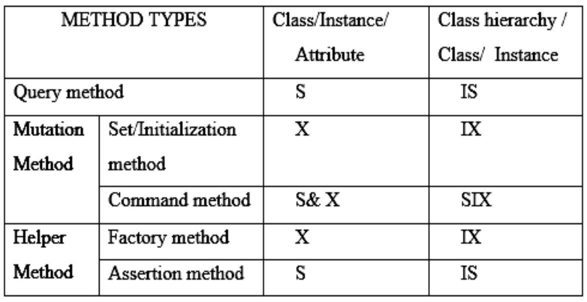
Instance objects can have only S and X locks. The class objects can be locked in S, X, IS, IX and SIX modes. The semantics of these modes are defined below:

- An IS (Intention Share) lock on a class means that instances of the class are to be explicitly locked in S or X mode as necessary.
- An IX (Intention Exclusive) lock on a class means instances of the class will be explicitly locked in S or X mode as necessary.
- An S (Shared) lock on a class means that the class definition is locked in S mode, and all instances of the class are implicitly locked in S mode and thus are protected from any attempt to update them.

- An SIX (Shared Intention Exclusive) lock on a class implies that the class definition is locked in S mode, and all instances of the class are implicitly locked in S mode and instances to be updated (by the transaction holding the SIX lock)will be explicitly locked in X mode.
- An X (Exclusive) lock on a class means that the class definition and all instances of the class may be read or updated.

Table 3 defines the locks based on the types of object methods. The types of locks are also based on the class level / instance level/ attribute level of access. For class level methods, the class hierarchy is locked by intension locks and classes are locked by S or X locks. If it is instance method, then class is set by intension locks and its instances are locked by S or X locks. The objects are accessible only after their creation. Their accessibility ceases after destruction.

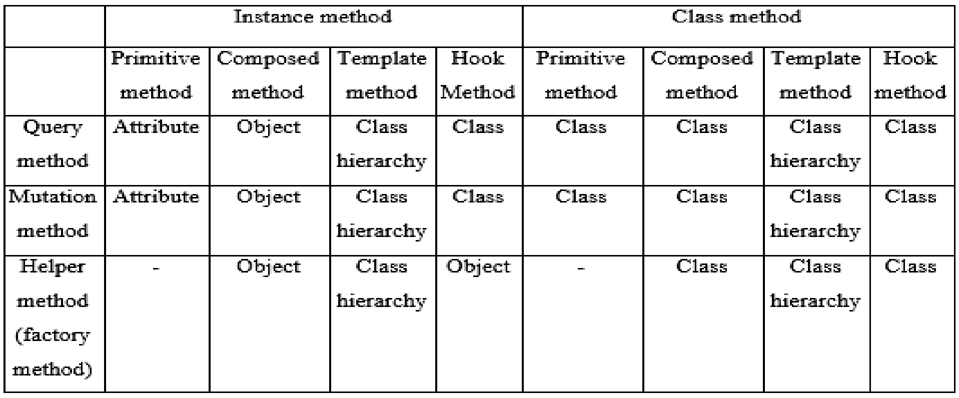
Table 3: Lock type based on method types of Inheritance



METHOD TYPES		Class/Instance/ Attribute	Class hierarchy / Class/ Instance
Query method		S	IS
Mutation Method	Set/Initialization method	X	IX
	Command method	S& X	SIX
Helper Method	Factory method	X	IX
	Assertion method	S	IS

Table 4 defines the lockable granules for various methods based on their properties as below. By combining the types and properties of the methods, the lock type and lockable granule size can be deduced.

Table 4: Lock granularity based on method properties in Inheritance



	Instance method				Class method			
	Primitive method	Composed method	Template method	Hook Method	Primitive method	Composed method	Template method	Hook method
Query method	Attribute	Object	Class hierarchy	Class	Class	Class	Class hierarchy	Class
Mutation method	Attribute	Object	Class hierarchy	Class	Class	Class	Class hierarchy	Class
Helper method (factory method)	-	Object	Class hierarchy	Object	-	Class	Class hierarchy	Class

2.4 Types and Granularity of Lock Based on Method Types for Aggregation

Aggregation is an object relationship. In order to maintain consistency, when a client requests a composite object, intension lock must be set on its class. Along with that, the component objects that constitute the composed object must also be set on intention object lock. These intention locks, while locking the particular object that constitute the composite objects, let other objects of the same class to be used by other clients. This improves concurrency. Aggregation may have exclusive or shared reference. Exclusive reference does not allow the component objects to be shared by other composite objects whereas shared reference allows it. Further aggregation may be dependent or independent on component objects for creation and deletion i.e. in the case of dependent aggregation, the composite object can be created only after creating the component objects and it is destroyed when all its composite objects are destroyed.

Table 5: Lock type based on method types for Aggregation

METHOD TYPES		Aggregation Root Object/ Attribute	Aggregation Root Class/ Object	Exclusive Component Class/Object	Shared Component Class/Object
Query method		S	IS/ISO	ISO/ISA	ISOS/ISAS
Mutation method	Set method/ Initialization method	X	IX/IXO	IXO/IXA	IXOS/SIXAS
	Command method	S/X	SIX/SIXO	SIXO/SIXA	SIXOS/SIXAS
Helper Method	Factory method	As per creation and deletion rule based on dependent / independent aggregation			
	Assertion method	S	IS/ISO	ISO/ISA	ISOS/ISAS

In the case of independent aggregation, the life cycle of composite object is independent of its composite objects. Table 5 gives the types of locks based on method types for aggregation. It is followed by granularity of locks as in Table 6.

Table 6: Lock granularity for Aggregation or Composition

Class Type	Granularity of locks	
	Primitive method	Composed method
Primitive class	Component attribute	Component object
Non Primitive class	Composite object hierarchy	

Table 7: Type of locks based on method types for Association

METHOD TYPES		Association Root Object/ Attribute	Association Root Class/ Object	Exclusive Associated Class/Object	Shared Associated Class/Object
Query method		S	IS/ISO	ISO/ISA	ISOS/ISAS
Mutation method	Set method / Initialization method	X	IX/IXO	IXO/IXA	IXOS/IXAS
	Command method	S/X	SIX/SIXO	SIXO/SIXA	SIXOS/SIXAS
Helper method	Factory method	As per creation and deletion rule basd on dependent / independent association			
	Assertion method	S	IS/ISO	ISO/ISA	ISOS/ISAS

Association is also an object relationship. In order to maintain consistency, when a client requests an associative object, intension lock must be set on its class. Further, the associated objects that constitute the associative object must also be set on intention object lock. These intention objects, while locking the particular object that constitute the associative objects, lets other objects of the same class to be used by other clients. This improves concurrency.

Association may have exclusive or shared reference. Exclusive reference does not allow the associated objects to be shared by other associative objects whereas shared reference allows it; further association may be dependent or independent on associated objects for creation and deletion. i.e. in the case of dependent association, the associative object can be created only after creating associated objects and it is destroyed when all its associated objects are destroyed. In the case of independent association, the life cycle of associative object is independent of its associated objects. Association relationship also possesses association hierarchy like aggregation hierarchy. Table 7 gives the types of lock based on method types for association. It is followed by granularity of locks as in Table 8.

Table 8: Lock granularity for Association

Class type	Granularity of locks	
	Primitive method	Composed method
Primitive class	Associated attribute	Associated object
NonPrimitive class	Associative object hierarchy	

2.5 Compatibility Matrix for Runtime Transactions Based on Class Relationships

The compatibility matrix specified in [6] for inheritance is given in table 9. The inheritance can be classified as exclusive inheritance or shared inheritance. The inheritance types single inheritance, multilevel inheritance, multiple inheritance allow exclusive inheritance of a parent class to one or more child classes. But in hierarchical inheritance, several sub classes are inherited from the same parent class or the parent is shared by many siblings.

Table 9: Compatibility matrix for Inheritance [6]

	IS	**IX**	**S**	**SIX**	**X**
IS	Y	Y	Y	Y	N
IX	Y	Y	N	N	N
S	Y	N	Y	N	N
SIX	Y	N	N	N	N
X	N	N	N	N	N

If the compatibility matrix specified in [6] is extended for this shared inheritance, then concurrency will be restricted. At any time, only one sub class is allowed to lock the parent class. Hence separate intension lock modes must be defined to increase concurrency. In the compatibility matrix below, separate lock modes need to be defined in shared and exclusive inheritance. Three more lock modes ISCS (Intension Shared Class Shared), IXCS (Intension Shared Exclusive Shared) and SIXCS (Shared Intension Exclusive Class Shared) can be defined to support shared inheritance. These new lock modes can be appended to the compatibility matrix in [6] as in table 10. Figures 8a, 8b and 8c show the different types of shared and exclusive inheritance and the locking policy in each type of inheritance. Table 10 gives the revised compatibility matrix.

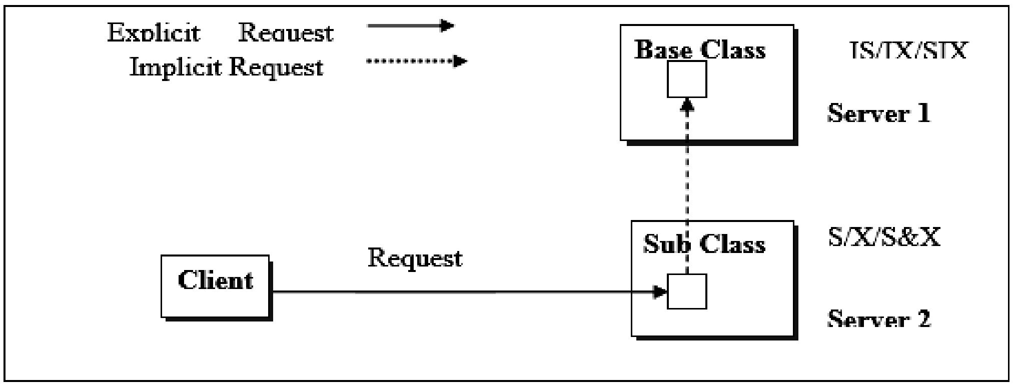


Fig. 8a: Locking in Single Inheritance

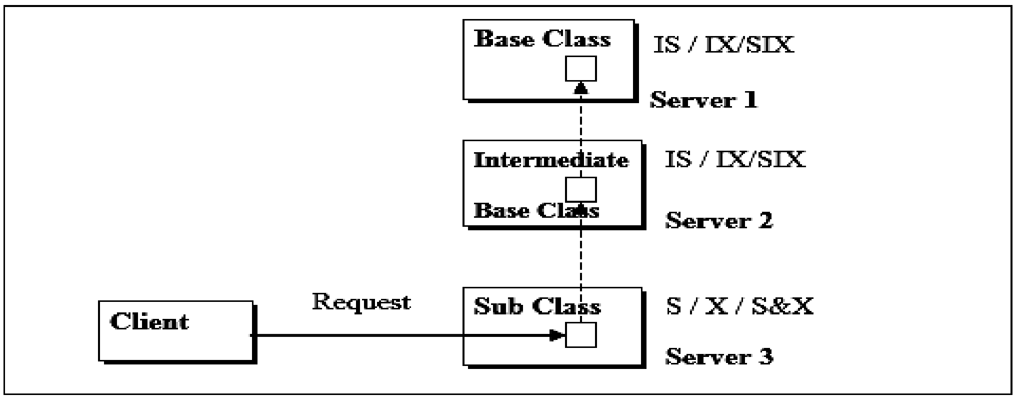


Fig. 8b: Locking in Multilevel Inheritance

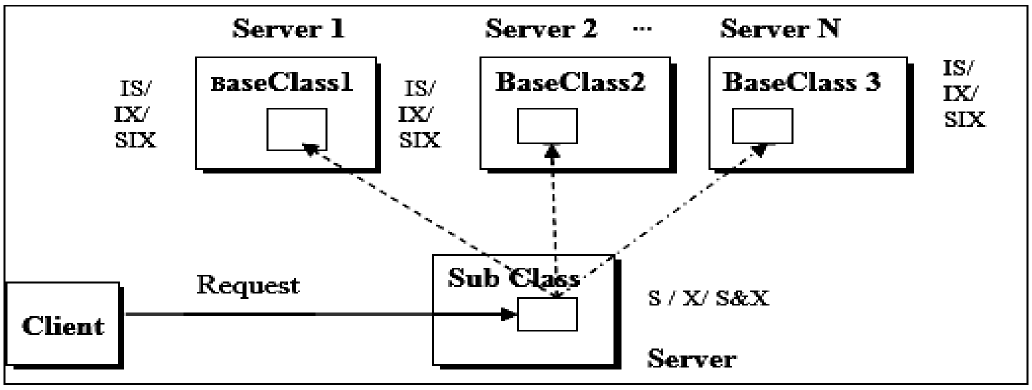


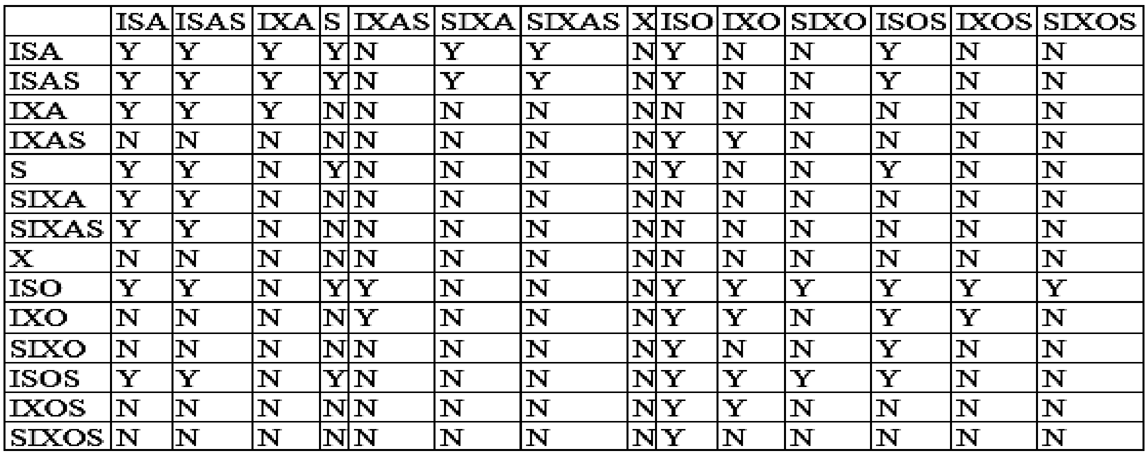
Fig. 8c: Locking in Multiple Inheritance

Table 10: Revised compatibility matrix for Inheritance

	IS	ISCS	IX	IXCS	S	SIX	SIXCS	X
IS	Y	Y	Y	Y	Y	Y	Y	N
ISCS	Y	Y	Y	Y	Y	Y	Y	N
IX	Y	Y	Y	Y	N	N	N	N
IXCS	Y	Y	Y	Y	N	N	N	N
S	Y	Y	N	N	Y	N	N	N
SIX	Y	Y	N	N	N	N	N	N
SIXCS	Y	Y	N	N	N	N	N	N
X	N	N	N	N	N	N	N	N

Table 11 gives the compatibility matrix for aggregation extended from [5]. In [5], compatibility matrix for aggregation has been defined by extending the lock modes defined for inheritance to aggregation. But its granularity size is restricted to object level. It is further extended to attribute level in the proposed scheme to improve the concurrency.

Table 11: Revised compatibility matrix for object relationships



	ISA	ISAS	IXA	S	IXAS	SIXA	SIXAS	X	ISO	IXO	SIXO	ISOS	IXOS	SIXOS
ISA	Y	Y	Y	Y	N	Y	Y	N	Y	N	N	Y	N	N
ISAS	Y	Y	Y	Y	N	Y	Y	N	Y	N	N	Y	N	N
IXA	Y	Y	Y	N	N	N	N	N	N	N	N	N	N	N
IXAS	N	N	N	N	N	N	N	N	Y	Y	N	N	N	N
S	Y	Y	N	Y	N	N	N	N	Y	N	N	Y	N	N
SIXA	Y	Y	N	N	N	N	N	N	N	N	N	N	N	N
SIXAS	Y	Y	N	N	N	N	N	N	N	N	N	N	N	N
X	N	N	N	N	N	N	N	N	N	N	N	N	N	N
ISO	Y	Y	N	Y	Y	N	N	N	Y	Y	Y	Y	Y	Y
IXO	N	N	N	N	Y	N	N	N	Y	Y	N	Y	Y	N
SIXO	N	N	N	N	N	N	N	N	Y	N	N	Y	N	N
ISOS	Y	Y	N	Y	N	N	N	N	Y	Y	Y	Y	N	N
IXOS	N	N	N	N	N	N	N	N	Y	Y	N	N	N	N
SIXOS	N	N	N	N	N	N	N	N	Y	N	N	N	N	N

The compatibility matrix for association has to give separate lock modes for attribute level association and object level association. Association is also an object relationship like aggregation. As lock modes for object level locking and attribute level locking has already been defined for aggregation, it can also be extended to association. Hence it is same as the compatibility matrix for aggregation as given in table 11. The compatibility matrix of table 12 completely defines the semantics of all the lock modes for run time transactions. It combines the compatibility matrix defined for each relationship separately as given in table 10 and 11.

2.6 Compatibility Matrix for Runtime and Design Time Transactions

In OODBMS, fine level lock modes are also defined for design time operations. It is not possible to extend the same to OODS, because it has no schema and query language support. When any design time operations are performed, the code implementing the domain has to be changed. As it is very difficult to predict which part of the code is getting modified in OODS, coarse level locking is offered for design time operations in OODS.

Table 12: Compatibility matrix for runtime transactions

	IS	ISCS	IX	IXCS	S	SIX	SIXCS	X	ISO	IXO	SIXO	ISOS	IXOS	SIXOS	ISA	IXA	SIXA	ISAS	IXAS	SIXAS
IS	Y	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N
ISCS	Y	Y	Y	Y	Y	Y	Y	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N
IX	Y	Y	Y	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
ISCS	Y	Y	Y	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
S	Y	Y	N	N	Y	N	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N
SIX	Y	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
SIXCS	Y	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
X	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
ISO	Y	Y	N	N	Y	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
IXO	N	N	N	N	N	N	N	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
SIXO	N	N	N	N	N	N	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N
ISOS	Y	Y	N	N	Y	N	N	N	Y	Y	Y	Y	N	N	Y	Y	Y	Y	N	N
IXOS	N	N	N	N	N	N	N	N	Y	Y	N	N	N	N	Y	Y	N	N	N	N
SIXOS	N	N	N	N	N	N	N	N	Y	N	N	N	N	N	Y	N	N	N	N	N
ISA	Y	Y	N	N	Y	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
IXA	N	N	N	N	N	N	N	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
SIXA	N	N	N	N	N	N	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N
ISAS	Y	Y	N	N	Y	N	N	N	Y	Y	Y	Y	N	N	Y	Y	Y	Y	N	N
IXAS	N	N	N	N	N	N	N	N	Y	Y	N	N	N	N	Y	Y	N	N	N	N
SIXAS	N	N	N	N	N	N	N	N	Y	N	N	N	N	N	Y	N	N	N	N	N

Table 13: Revised compatibility matrix for design time transactions and runtime transactions

	RD	WD	RA
RD	Y	N	Y
WD	N	N	N
RA	Y	N	Depends on table 12

The schema locking defined in Lee1996 may be taken into account for design time transactions. Just as schemas are changed periodically, OODS can also provide improved services. This requires updating of behavior defined by object methods. The lock modes can be called as RD (Read Definition), WD (Write Definition) and RA (Runtime Access). Then analogous to the schema locks RS and WS, compatibility matrix can be defined. The compatibility matrix for design time transactions and runtime transactions are defined in table 13.

III. Conclusion

The compatibility matrix mentioned in this chapter needs to be implemented. In OODBMS, the compatibility matrix is implemented as part of the DBMS. In OODS the domain is implemented using object oriented languages like java, c++ etc. Then it has to be implemented as operating system services or language constructs in programming languages say as an extended library of the language or as part of the component itself. Providing concurrency control at operating system level is too complex. Providing it at language level is possible. Already Java, Eiffel etc., offers such extended libraries for various services. Among all the solutions, implementing it as part of the component is much more feasible. Already, COM has set a precedence of managing the clients in a primitive way using reference counts. Active component approach called JADEX has been proposed in [15], in which the concurrency module is built as part of the component. They have provided primitive concurrency control mechanism to avoid dirty reads and writes. They have not exploited the semantics of object oriented paradigm. If the proposed compatibility matrix can be incorporated in such components the performance will improve.

This paper proposes a semantic based concurrency control mechanism for object oriented distributed systems. It is based on multi granular lock model. The Compatibility matrix defines lock modes for objects based on the semantics of object oriented paradigm. It provides fine granularity for runtime data requests. But design time requests are still in coarse level. Hence the future work will be to explore the possibility of providing fine granularity also for design time requests.

References

[1] V.Geetha and N.Sreenath, "Impact of Object Operations and Relationships in Concurrency Control in DOOS", International Conference on Distributed Computing and Networking, Kolkata, Proceedings in Springer-Verilag, 2010.

[2] Dirk Riehle, Stephen P. Berczuk, "Types of Member Functions in C++", Report, 2000.

[3] Dirk Riehle, Stephen P. Berczuk, "Properties of Member Functions in C++", Report, 2000.

[4] Shengli Wu and Nengbin Wang, "Directed Graph based Association Algebra for Object Oriented Database", IEEE, pp 53- 59, 1998.

[5] W.Kim, E.Bertino and J.F.Garza, "Composite Objects revisited," Object oriented Programming, systems, Languages and Applications, pp 327-340, 1990.

[6] J.F. Garza and W.Kim,"Transaction management in an object oriented database system", Proc. ACM SIGMOD Int'l conference, management data, 1987.

[7] Dragan Milicev, "On the Semantics of Associations and Association Ends in UML", IEEE Transactions on Software Engineering, Vol.33, No.4, pp 238-251, April 2007.

[8] Brian Henderson-Sellers, "Towards the formalization of Relationships for Object Modeling", Centre for Object Technology Applications and Research.

[9] P.Stevens, "On the Interpretation of Binary Associations in the Unified Modeling Language ", vol. 1, No. 1, pp68-79, 2002.

[10] Tom Pender, "UML 2 Bible", Wiley Publishing Inc., First Edition, 2003.

[11] S.Y.Lee and R.L. Liou, " A Multi- Granularity Locking model for concurrency control in Object–Oriented Database Systems", IEEE Transactions on Knowledge and Data Engineering, Vol 8, no 1, feb 1996.

[12] Woochun Jun," A multi- granularity locking-based concurrency control in object oriented database system, Elsevier Journal of Systems and Software, pp 201-217, 2000.

[13] Woochun Jun and Le Gruenwald, "An effective class hierarchy concurrency control technique in object–oriented database systems", Elsevier Journal of Information and Software Technology, pp 45-53, 1998.

[14] J.N.Gray, R.A. Lorie, G.R. Putzolu and L.I. Traiger, "Granularity of locks and degrees of consistency in shared database," Modeling in Database management system, G.M. Nijssen, ed, Elsevier, North Holland, pp 393-491, 1978.

[15] L. Braubach and A.Pokahr, "Intelligent Distributed Computing V", Proceedings of the 5PthP International Symposium on Intelligent Distributed Computing (IDC 2011), Springer, pp141-151, 2011.

Authors' Profiles

V.Geetha: Assistant Professor (Senior) in Information Technology department, Pondicherry engineering college. Currently she is doing her Ph.D in Pondicherry University. Her research areas of interest includes client/ server architecture, distributed systems, object oriented systemdesign and middleware technologies.

N.Sreenath: Professor in Department of Computer Science and Engineering in Pondicherry Engineering College. He has more than 30 publications in various international conference proceedings and journals. His areas of interest are distributed computing, high speed networks and WDM optical networks.

How to cite this paper: V.Geetha, N.Sreenath,"Semantic Multi-granular Lock model for Object Oriented Distributed Systems", International Journal of Information Technology and Computer Science(IJITCS), vol.5, no.5, pp.74-84, 2013.DOI: 10.5815/ijitcs.2013.05.10