QoS-based Hybrid Concurrency Control on Distributed Multimedia Objects

Tomoya Enokido and Makoto Takizawa
Dept. of Computers and Systems Engineering
Tokyo Denki University, Japan
{eno, taki}@takilab.k.dendai.ac.jp

Abstract

Distributed multimedia objects are concurrently manipulated by multiple transactions. We define new types of conflicting and exclusive relations on methods to serialize transactions and to realize exclusive manipulation of an object by taking into account change of QoS and state. We discuss a locking protocol for realizing mutual exclusion on objects and a scheduler for serializing transactions.

1. Introduction

In various kinds of distributed applications, multimedia objects like video and voice are concurrently manipulated by multiple transactions. Multimedia objects are larger and structured in a part-of relation. The traditional concurrency control protocols [1, 5] cannot be adopted for multimedia objects due to the low throughput. In addition, it is significant to discuss what quality of service (QoS) like frame rate and number of colours each multimedia object supports for applications [6]. Each object is an encapsulation of state, i.e. data and methods for manipulating the state. The object can be manipulated only through the methods. There are a pair of aspects to discuss properties of methods on multimedia objects; state and QoS types. State and QoS of object are manipulated by state and QoS methods, respectively. Nemoto et al. [10] discuss novel types of conflicting relations among methods on the basis of state and QoS of object. In this paper, we extend and refine types of conflicting relations in order to make the meanings of methods more strict. Based on the conflicting relation, the computation order of conflicting methods are decided.

According to the traditional concurrency control theories [1, 5], every pair of conflicting methods cannot be concurrently performed, while every pair of compatible methods can be concurrently performed. In this paper, we newly discuss QoS-based exclusive relations, which show whether or not a pair of methods can be concurrently performed on an object. We discuss a hybrid type of concurrency control with locking protocol and timestamp ordering (TO) scheduler to perform multiple transactions manipulating multimedia objects distributed on multiple servers. Here, method requests from multiple transactions are scheduled to be serialized by the scheduler according to the QoS-based conflicting relations of methods in each server. Then, methods are performed if the methods are not mutually exclusive with methods being concurrently performed according to the QoS-based mutual exclusive relation.

In section 2, we discuss consistent relations among object states. In section 3, we discuss conflicting and exclusive relations among methods. In section 4, we discuss the concurrency control algorithms. Section 5 concludes the papers.

2. System Model and Definitions

2.1 System model

A class c is composed of attributes and methods. An object o is an instantiation of the class c. A state of the object o is given in a tuple of attribute values. Each object has one state at a time. A class c can be composed of component classes c1, . . . , cn. Let cci denote a component class ci of the class c. Let ci(s) denote a projection of a state s of the class c to a component class ci. For example, a class karaoke is composed of three component classes music, words, and background [Figure 1]. Let op(s) and [op(s)] denote a state and represent the output obtained by performing a method op on a state s of the object o, respectively. Here, op1 ∪ op2 and op1 ∥ op2 denote that op1 and op2 are serially and concurrently performed, respectively.

Figure 1. Karaoke class.

Applications obtain service from multimedia objects only through methods supported by objects. Each service is characterized by quality of service (QoS) like level of resolution. Each state s of an object o supports QoS denoted by Q(s). Q(s1) dominates Q(s2) (Q(s1) ⊇ Q(s2)) if a state s1 supports better QoS than another state s2. For example, {30[fps], 16[colours]} ⊇ {40[fps], 64[colours]}. Let Q be a set of all possible QoS instances. The set Q with the dominant relation ⊇ is a lattice (Q, ≤, ∪, ∩). A least upper bound (lub) q1 ∪ q2 of QoS instances q1 and q2 is some q3 in Q such that q1 ≤ q3, q2 ≤ q3, and no q4 in Q where q1 ≤ q4 ≤ q3 and q2 ≤ q4 ≤ q3. A greatest lower bound (glb) q1 ∩ q2 is a QoS instance q3 such that q3 ≤ q1, q3 ≤ q2, and no QoS instance q4 where q3 ≤ q4 ≤ q1 and q3 ≤ q4 ≤ q2. An application requires an object to support requirement of service (RoS).

2.2 Definitions

A karaoke object k1 is composed of a music object m1, words object w1, and background object b1. Another karaoke object k2 is same as k1 except that the background object is b2 (b1 ≠ b2). An application considers a pair of the objects k1 and k2 to be consistent since the application is interested in only words and music.

If the application is interested in state and QoS of a component class c, c is state (S) and QoS (Q) significant, respectively. A class c is referred to as SQ. S, and Q-type if
the class c is S and Q, S, and Q-significant, respectively. In addition, if the class c is not significant, c is referred to as NSignificant. Let stype(c) denote a significant type of a class c. There are following types of consistent relations between a pair of states s_t and s_u of a class c. Here, r shows request QoS (RoS).

- s_t is state and QoS (SQ) consistent with s_u \((s_t \sim s_u)\) iff \(s_t \sim s_u\).
- s_t is state (S) consistent with s_u \((s_t \sim s_u)\) iff \(s_t \sim s_u\) and s_u are obtained by degrading QoS of some state s of c.
- s_t is QoS (Q) consistent with s_u \((s_t \sim s_u)\) and \(Q(s_t) \equiv Q(s_u)\).
- s_t is semantically SQ (SemSQ) consistent with s_u \((s_t \equiv s_u)\) iff \(s_t \sim s_u\) or \(c_i(s_t) \equiv c_i(s_u)\) for every SQ-type component class c_i of c.
- s_t is semantically S (SemS) consistent with s_u \((s_t \simeq s_u)\) iff \(s_t \sim s_u\ or c_i(s_t) \simeq c_i(s_u)\) for every SQ or S-type component class c_i of c.
- s_t is semantically Q (SemQ) consistent with s_u \((s_t \preceq s_u)\) iff \(s_t \sim s_u\ or c_i(s_t) \preceq c_i(s_u)\) for every semantically consistent class c_i of c.
- s_t is state and RoS r \((S[r])\) consistent with s_u \((s_t \sim [r] s_u)\) iff \(s_t \sim s_u\ and Q(s_t) \cap Q(s_u) \geq r\).
- s_t is \([r]\) consistent with s_u \((s_t \equiv [r] s_u)\) iff \(s_t \sim s_u\ and Q(s_t) \cap Q(s_u) \geq r\).
- s_t is semantically \([r]\) (Sem\([r]\)) consistent with s_u \((s_t \equiv [r] s_u)\) iff \(s_t \sim s_u\ or c_i(s_t) \equiv c_i(s_u)\) for every Q-type component class c_i.
- s_t is semantically \([r]\) (Sem\([r]\)) consistent with s_u \((s_t \equiv [r] s_u)\) iff \(s_t \sim s_u\ or c_i(s_t) \equiv c_i(s_u)\) for every Q-type component class c_i.

In Figure 1, suppose an image object is fully colored and a sound object supports a stereo type of sound. A state s_1 is obtained by changing a state s of the car object with monochromatic image. A state s_2 is obtained by changing the sound object with monochromatic one. Here, s_1 and s_2 are S-consistent \((s_1 \sim s_2)\) but s_1 \(\not\sim s_2\). Thus, s_3 is S-consistent with s_u but Q(s_3) \(\not\sim s_u\) may not be the same.

For a set R of all possible RoS instances, let \(\lnot R \equiv R\), and \(\lnot \lnot R \equiv \lnot R\) denote collections of consistent relations, \{\lnot \lnot r \mid r \in R\}, \{\equiv r \mid r \in R\}, \{\lnot \lnot r \mid r \in R\}, and \{\lnot \lnot r \mid r \in R\} which are referred to as SR, semantically SR(SemSR), R, and semantically \(R(SemR)\) consistent relations, respectively. Here, let SQ, S, Q, SemSQ, SemS, SemQ, SR, R, SemSR, and SemR denote consistency sets of possible SQ, S, Q, SemSQ, SemS, SemQ, SR, R, SemSR, and SemR consistent relations of a class c. Let C be a consistency family \(\{SQ, S, Q, SemSQ, SemS, SemQ, SR, R, SemSR, SemR\}\) of the consistent relations. Let \(\sqcup\) denote a \(\alpha\)-consistent relation for \(\alpha \in C\). For a pair of methods op_1 and op_2 of a class c, "op_1 \(\sqcup\) op_2" denotes that "op_1(s) \(\sqcup\) op_2(s)" holds for every state s of a class c. For \(\alpha\) and \(\beta\) in C, "\(\alpha\) dominates \(\beta\)" \((\alpha \rightarrow \beta)\) means \(\alpha \leq \beta\), denoting that \(s_t \sqsubseteq s_u\) if \(s_t \sqsubseteq s_u\) for every pair of states s_t and s_u of a class c. Figure 2 denotes a Hasse diagram where a node \(\alpha\) denotes a consistency set \(\alpha\) in C and a directed edge from \(\alpha\) to \(\beta\) denotes "\(\alpha\) dominates \(\beta\)". For example, "SQ \(\rightarrow\) SemSQ" means that \(s_1 \equiv s_2\) if \(s_1 \sqsubseteq s_2\) for every pair of states s_1 and s_2. For RoS instances r_1 and r_2 in R, r_1 dominates r_2 \((r_1 \rightarrow r_2)\) iff \(r_1 \geq r_2\).

![Figure 2. Hasse diagram.](image)

### 3. Conflicting and Exclusive Relations

#### 3.1 \(\alpha\)-conflicting relations

A method op_\(\alpha\) is \(\alpha\)-compatible with a method op_\(\beta\) for a consistency set \(\alpha\) in the consistency family C \((op_\alpha \ominus op_\beta)\) iff \(op_\alpha\) and \(op_\beta\) commute under \(\alpha\), i.e. \((op_\alpha \circ op_\beta(s)) \sqsubseteq \alpha\) \((op_\beta \circ op_\alpha(s))\) for every state s of a class c. A method \(op_\alpha\) \(\alpha\)-conflicts with \(op_\beta\) \((op_\alpha \circ \ominus \alpha \cdot op_\beta)\) iff \(op_\alpha\) is not \(\alpha\)-compatible with \(op_\beta\). Here, the compatible relation \(\alpha\) is assumed to be symmetric but not transitive. Let \(\alpha\) and \(\beta\) represent a pair of consistency sets in C. A pair of methods \(op_\alpha\) and \(op_\beta\) are allowed to be performed in any order for \(\alpha\) in C if \(op_\alpha \ominus op_\beta\). It is straightforward that \(\alpha\) \(\subseteq\) \(\beta\) iff \(\alpha \leftarrow \beta\) for every \(\alpha\) and \(\beta\) in C.

A Hasse diagram isomorphic with Figure 2 holds for the compatible and conflicting relations from the theorem.

#### 3.2 \(\alpha\)-serializability

Let \(T_i\) and \(T_j\) be a pair of transactions issuing methods \(op_\alpha\) and \(op_\beta\) to an object \(\alpha\), respectively. Some consistency set \(\text{ctype}(T_i) \in C\) is given to each transaction \(T_i\) which is decided by an application. A transaction \(T_i\) \(\alpha\)-precedes the another transaction \(T_j\) with respect to a consistency set \(\alpha\) in C \((T_i \preceq T_j)\) iff \(op_\alpha\) is started before \(op_\beta\) on the object \(\alpha\) and \(op_\alpha\) \(\alpha\)-conflicts with \(op_\beta\) \((op_\alpha \circ \ominus \alpha \cdot op_\beta)\) for \(\alpha \geq \text{ctype}(T_i) \cup \text{ctype}(T_j)\). A transaction \(T_i\) \(\alpha\)-precedes another transaction \(T_j\) with respect to a consistency set \(\alpha\) \((T_i \preceq T_j)\) iff \(T_i \preceq T_j\) or \(T_i \preceq T_j \preceq T\) for \(\alpha \geq \alpha_1 \cup \alpha_2\).

**[Theorem]** A transaction \(T_i\) \(\alpha\'-\precedes\) another transaction \(T_j\) \((T_i \preceq T_j)\) if \(T_i \preceq T_j\) and \(\alpha \rightarrow \alpha'\) for a pair of consistency sets \(\alpha\) and \(\alpha'\).

**[\(\alpha\)-serializability]** A collection T of transactions \(T_1, \ldots, T_m\) is \(\alpha\)-serializable with respect to a consistency set \(\alpha\) if both \(T_i \preceq T_j\) and \(T_j \preceq T_i\) do not hold for every pair of transactions \(T_i\) and \(T_j\) where \(\alpha \geq \text{ctype}(T_i) \cup \text{ctype}(T_j)\), i.e., \(\preceq\) is acyclic.

Let T be a set of \(\{T_1, \ldots, T_n\}\) of transactions \((n \geq 1)\). T is minimally \(\alpha\)-serializable if T is \(\alpha\)-serializable but not \(\alpha\)-serializable for every consistency set \(\alpha\) in C such that \(\alpha \rightarrow \alpha'\). For example, even if transactions are SemSQ-serializable, the transactions may not be SQ-serializable.

transaction sets $\alpha_1$ and $\alpha_2$ in $\text{SR}$, “$\alpha_1 \rightarrow \alpha_2$” denotes $\alpha_1 \subseteq \alpha_2$. Suppose $\alpha_1 \rightarrow \alpha_2$ for a pair of transaction sets $\alpha_1$ and $\alpha_2$ in $\text{SR}$. This means, $T$ is $\alpha_2$-serializable if $T$ is $\alpha_1$-serializable. For example, $T$ is $Q$-serializable if $T$ is $S$-serializable. $\alpha_1 \rightarrow \alpha_2$ for a pair of transaction sets $\alpha_1$ and $\alpha_2$ in $\text{SR}$ if $\alpha_1 \rightarrow \alpha_2$ for a pair of consistency sets $\alpha_1$ and $\alpha_2$ in $C$. That is, the Hasse diagram isomorphic with Figure 2 holds for $\text{SR}$.

3.3 $\alpha$-exclusive relations

Let $\{op_1 \parallel op_a(s)s\}$ be a set of possible states obtained by concurrently performing a method of $op_1$ and $op_a$ on a state $s$ of a class $c$. A pair of methods $op_1$ and $op_a$ are $\alpha$-exclusive for a consistency set $\alpha$ if $op_1 \parallel op_a$ iff $s' \not\sqsubseteq (op_1 \circ op_a(s))$ or $s' \not\sqsubseteq (op_2 \circ op_a(s))$ for every state $s$ and every state $s' \in \{op_1 \parallel op_a(s)\}$.

A pair of methods $op_1$ and $op_a$ are $\alpha$-exclusive if $op_1 \parallel op_a$ does not hold. If $op_1 \parallel op_a$, $op_1$ and $op_a$ cannot be concurrently performed with respect to the consistency set $\alpha$. Let $A$ be the set of $\alpha$-exclusive relations of methods for every $\alpha$ in $C$. $\alpha$ denotes a set of $\alpha$-exclusive relations of methods where $\alpha \in \{\text{SQ, S, Q, } \text{SemSQ, SemS, SemQ, SR, R, SemSR, SemR}\}$. A Hasse diagram isomorphic with Figure 2 holds for $A$.

4. Hybrid Concurrency Control

4.1 Timestamp ordering (TO) scheduler

A transaction in a client issues method $op$ to a server computer to manipulate an object $o$. Each object is concurrently manipulated by multiple transactions. Each server supports a scheduler and locking protocol [1] [Figure 3]. Each transaction is associated with a consistency set $\alpha$ in the consistency family $C$. The scheduler is used to serialize conflicting methods with consistency sets in $C$. An object is locked in order to realize the exclusive relations among methods. Each transaction $T$ is assigned a unique timestamp $ts(T)$ [1] which denotes a local time and identifier of client when and where $T$ is initiated, respectively [1]. Every method $op$ issued by a transaction $T$ carries the timestamp $ts(T)$, i.e. $ts(op) = ts(T)$. We assume each transaction issues a method by using a synchronous remote procedure call.

Suppose the transaction $T$ with some consistency set $\alpha$ issues a method $op$ to an object $o$. Here, $\text{ctype}(op) = \text{ctype}(T)$. Multiple transactions issue methods to the TO scheduler of an object $o$. The methods issued are buffered in the scheduler and are ordered according to the following timestamp ordering rule:

[Consistent timestamp ordering (CTO) rule] A method $op_1$ precedes another method $op_2$ in the scheduler ($op_1 \Rightarrow op_2$) iff $ts(op_1) < ts(op_2)$ and $op_1$ $\alpha$-conflicts with $op_2$ ($op_1 \not\sqsubseteq (op_1 \circ op_2)$ or $op_2 \not\sqsubseteq (op_1 \circ op_2)$) for every $\alpha \geq \text{ctype}(op_1) \cup \text{ctype}(op_2)$.

Methods in the scheduler are totally ordered in CTO rule. A method $op_1$ directly precedes another method $op_2$ (or $op_2$ directly follows $op_1$) iff $op_1 \Rightarrow op_2$ and there is no method $op_3$ such that $op_1 \Rightarrow op_3 \Rightarrow op_2$ in the scheduler.

A pair of transactions $T_1$ and $T_2$ where $ts(T_1) < ts(T_2)$ issue methods $\text{gray-scale}$ and $\text{add-car}$ to a background object $b$, respectively. Suppose $\text{ctype}(T_1) = \text{ctype}(T_2) = Q$. Since $\text{gray-scale}$ $Q$-conflicts with $\text{add-car}$ ($\text{gray-scale} \not\sqsubseteq \text{add-car}$), $T_1$ $Q$-conflicts with $T_2$. $\text{gray-scale} \Rightarrow \text{add-car}$ since $ts(\text{gray-scale}) < ts(\text{add-car})$. Next, suppose $\text{ctype}(T_1) = Q$ and $\text{ctype}(T_2) = [r]$ where $[r]$ denotes “application is not interested in the colour of a car.” $Q \cup [r] = [r]$ since $\text{gray-scale} \not\sqsubseteq [r]$. Hence, $\text{add-car} \Rightarrow \text{gray-scale}$ even if $ts(\text{gray-scale}) < ts(\text{add-car})$.

[Figure 3. Scheduler and locking.]

Suppose there are a pair of transactions $T_1$ and $T_2$ which manipulate a background object $b$ where $ts(T_1) < ts(T_2)$. $T_1$ issues a method $\text{gray-scale}$ and $T_2$ issues $\text{add-car}$ to the object $b$. Suppose $\text{ctype}(T_1) = Q$ and $\text{ctype}(T_2) = S$. $S = Q \cup [r]$ Since $\text{gray-scale} \not\sqsubseteq [r]$, $\text{add-car}$ is required to precede $\text{add-car}$ in the scheduler. Next, suppose $\text{ctype}(T_1) = \text{ctype}(T_2) = [r]$ (“not interested in colour of car”). Since $\text{gray-scale} \not\sqsubseteq [r]$, $\text{add-car}$ can precede $\text{gray-scale}$.

4.2 Locking protocol

Let $E_o$ denote a set of methods which are being concurrently performed on an object $o$. The method $op$ can be performed on the object $o$ if every method in $E_o$ is $\alpha$-compatible with $op$ with respect to some concurrency set $\alpha$ in $C$.

[Execution rule ER1] A method $op$ is performed on an object $o$ if one of the following conditions is satisfied:

1. $E_o$ is empty, $E_o = \phi$.
2. If $E_o \neq \phi$, the method $op$ is not $\alpha$-exclusive with every method $op'$ ($op' \parallel op'$) in $E_o$, for a consistency set $\alpha \geq \text{ctype}(op) \cup \{\text{ctype}(op') \mid op' \in E_o\}$.

If the top method $op$ in the scheduler does not satisfy the execution rule ER1, $op$ has to stay in the scheduler. Until the top method satisfies the rule ER1, every other method has to wait in the scheduler. A method $op_1$ is ready with another method $op_2$ in the scheduler if $op_1 \Rightarrow op_2$ satisfies the execution rule ER1 and $op_1$ is $\alpha$-compatible with $op_2$ ($op_1 \not\sqsubseteq (op_1 \circ op_2)$ for a consistency set $\alpha \geq \text{ctype}(op_1) \cup \text{ctype}(op_2)$).

Suppose a top method $op$ does not satisfy the execution rule ER1 in the scheduler, i.e. some method exclusive with $op$ is now being performed. A method which is $\alpha$-compatible with the top method $op$ and is preceded by $op$ in the scheduler can be performed by the following rule:

[Execution rule ER2]

1. while (the top method $op$ in the scheduler satisfies the execution rule ER1) do { $op$ is removed from the scheduler; $op$ is performed; }
2. \( op \) = top method in the scheduler; \( \forall op \) cannot be performed since the rule ER1 is not satisfied. */
3. if \( op \) is the last method in the scheduler, stop;
4. if (a method \( op' \) directly following the method \( op'' \) in the scheduler is ready with \( op' \), \( \{ op' \} \) is removed from the scheduler; \( op' \) is performed; go to 2) else stop.

Suppose there are three transactions \( T_1, T_2, \) and \( T_3 \) where \( ts(T_1) < ts(T_3) < ts(T_2) \) and a pair of objects \( x \) and \( y \) in servers \( v_1 \) and \( v_2 \), respectively. \( T_1 \) issues methods \( op_1 \) and \( op_2 \) to objects \( x \) and \( y \), respectively. \( T_2 \) issues methods \( op_3 \) and \( op_4 \) to objects \( x \) and \( y \), respectively. \( T_3 \) issues a method \( op_5 \) to the object \( y \) in the server \( v_2 \). Let \( S_1 \) and \( S_2 \) be schedulers of servers \( v_1 \) and \( v_2 \), respectively. The methods issued by \( T_1, T_2, \) and \( T_3 \) are stored in the schedulers \( S_1 \) and \( S_2 \) as shown in Figure 4. Suppose that \( op_1 \) \( \not\equiv op_2 \) but \( op_3 \equiv op_4 \). Suppose \( op_3 \not\equiv op_5 \). \( op_1 \) precedes \( op_2 \) \( (op_1 \Rightarrow op_2) \) in the scheduler \( S_1 \) and \( op_3 \Rightarrow op_4 \Rightarrow op_5 \) in \( S_2 \). Suppose \( op_1 \) satisfies the rule ER2. \( op_1 \) is taken from the scheduler \( S_1 \) and is performed on the object \( x \). Then, \( op_2 \) is examined for the rule ER2. Since \( op_2 \not\equiv op_1 \), \( op_1 \) is kept waited in \( S_1 \) until \( op_1 \) completes. Next, suppose \( op_3 \) satisfies the rule ER2. The method \( op_3 \) is taken from the scheduler \( S_2 \) and \( op_3 \) is performed on the object \( y \). Here, suppose \( op_3 \) cannot be performed since some method exclusive with \( op_3 \) is being performed. Since \( op_3 \) is SemS-compatible with \( op_4 \) \( \not\equiv op_5 \), \( op_4 \) and \( op_5 \) can be exchanged in \( S_2 \). If \( op_4 \) satisfies ER2, \( op_4 \) can be taken before \( op_5 \) and performed on the object \( y \). In the server \( v_1, T_1 \) \( \not\equiv \) preceding \( T_2 \) \( (T_1 \not\equiv T_2) \) since \( op_1 \) \( \not\equiv \) preceding \( op_2 \) \( (op_1 \not\equiv op_2) \) and \( op_1 \) is performed before \( op_2 \) in \( v_1 \). On the other hand, \( op_3 \) from \( T_2 \) is performed before \( op_4 \) from \( T_1 \) in \( v_2 \). However, \( op_5 \) and \( op_4 \) are SemSQ-compatible. Hence, \( T_1 \) and \( T_2 \) are SemSQ-separable.

In order to realize the rule ER2, the locking mechanism is adopted. Before performing a method \( op \) on an object \( o \), \( o \) is locked in a mode \( \mu(op) \). If \( op \) is an \( \alpha \)-exclusive with \( op_2 \), \( \mu(op_1) \) is an \( \alpha \)-exclusive with \( \mu(op_2) \). Suppose an object \( o \) is locked in \( \mu(op_2) \). If \( \mu(op_1) \) is an \( \alpha \)-exclusive with \( \mu(op_2) \), the object \( o \) cannot be locked in \( \mu(op_1) \) for \( \alpha = \) ctype \( (op_1) \cup \) ctype \( (op_2) \).

For a top method \( op \) in the scheduler, a lock request of a mode \( \mu_\alpha(op) \) is issued to an object \( o \) where \( \alpha \equiv \) ctype \( (op) \). For every method \( op' \) in the \( E_o \), if \( \mu_\alpha(op') \) is not \( \alpha'' \)-exclusive with \( \mu_\alpha(op) \) for a consistency set \( \alpha'' \supseteq \alpha' \cup \alpha \), the object \( o \) is locked in \( \mu_\alpha(op) \). If succeeded in locking \( o \), \( op \) is started performed. Here, \( mts(op) := \max(ts(op), mts(op)) \). Otherwise, \( op \) is kept waited in the scheduler. If a transaction terminates, locks held by the transaction are released.

5. Concluding Remarks

In this paper, we defined novel types of consistent relations among methods by taking into account QoS change in addition to state change of objects, i.e., the consistency family \( C = \{ SQ, S, Q, SemSQ, SemS, SemQ, SR, SemSR, SemR \} \). Based on a consistency set \( \alpha \) in \( C \), we defined the \( \alpha \)-conflicting and \( \alpha \)-exclusive relations among methods. We discussed the scheduler and locking protocols to serialize conflicting methods from multiple transactions and to perform methods in mutual exclusion.

References