社団法人 情報処理学会　研究報告
IPSJ SIG Technical Report
2003－DPS－115　(12)
2003／11／14

Concurrency Control Protocol on Distributed Multimedia Objects

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

Abstract

Multimedia objects distributed in networks are concurrently manipulated by multiple transactions like co-authoring systems. Multimedia objects are larger and structured in a part-of relation. Not only state but also quality of service (QoS) of object are changed through methods. We define new types of conflicting relations on methods by taking into account QoS while only state change is considered in traditional concurrency controls. We discuss a protocol for locking objects to be consistent with respect to the QoS-based conflicting relations.

分散マルチメディアオブジェクト上における同時実行制御プロトコル

蓮見 圭亮　榎戸 智也　滝沢 誠
東京電機大学大学院理工学研究科情報システム工学専攻
E-mail {hasu, eno, taki}@takilab.k.dendai.ac.jp

分散アプリケーションでは、複数のユーザによってマルチメディアオブジェクトが同時に操作される。オブジェクトは複数の副オブジェクトが part-of 関係によって階層構造になっている。また、各オブジェクトの状態と QoS(Quality of Service) が、メソッドにより操作される。本論文では、従来の同時実行制御の中で新たに QoS に基づいたメソッド間の競合関係を定義する。更に、QoS に基づいた競合関係を用いて一貫性を保つためのオブジェクトのロックプロトコルを提案する。

1 Introduction

In various kinds of distributed applications, multimedia objects like video and voice are manipulated. In co-authoring systems and cooperative working systems [1], multiple applications not only retrieve but also manipulate multimedia objects which are distributed in networks. There are many discussions about concurrency control on traditional data like two-phase locking [6] and timestamp ordering protocols [1]. Differently from traditional data, multimedia objects are larger and structured. In addition, it is significant to discuss what quality of service (QoS) like frame rate and number of colours each object supports for applications. Each object is an encapsulation of data and methods. 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. For example, the frame rate of a video is changed by *QoS* method while component objects are added to an object by *state* method. Some types of methods might change both state and QoS of object. The authors [14] 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 so that of methods to make the meanings of methods more strict. We also discuss the hybrid type of concurrency control with locking and timestamp ordering mechanisms to manage multiple transactions manipulating multimedia objects.

In section 2, we discuss a system model. In section 3, we discuss a hierarchical locking protocol.

2 Consistent Relations

A system is composed of *classes* and *objects*. A class c is composed of *attributes* and *methods*. An object o is an instanciation of the class c. A tuple of attribute values is a $state$ of the object o. Each object has one state at a time. A *state* of a class also means a state of the object. An object has a unique invariant identifier, i.e. object identifier (oid) while its state is variant.

A new class c_2 can be derived from an existing class c_1. In addition, a class c can be composed of $component$ classes $c_1, \ldots, c_n$. Let $c.c_i$ show a component class c_i of the class c. Let $c_i(s)$ denote a projection of a state s of the class c to a component class c_i. For example, a class *karaoke* [Figure 1] is composed of three component classes, $music$, $words$, and *background* [Figure 2]. *background*(*k*) shows a state of *background* in a state *k* of a *karaoke* object. The *background* class is furthermore composed of *car*, $tree$, and $cloud$ classes.

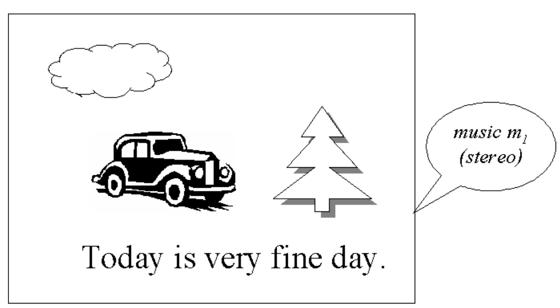

Figure 1. Karaoke object.

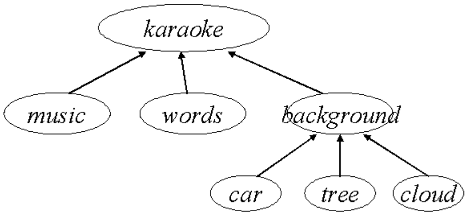

Figure 2. Karaoke class.

Let $op(s)$ denote a state obtained by performing a method op on a state s of the object o. Let [$op(s)$] show output obtained by performing a method op on the state s. Here, $op_1 \circ op_2$ and $op_1 \oplus op_2$ show that a pair of methods op_1 and op_2 are serially and concurrently performed, respectively.

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). For example, *Q*($play$(s)) shows QoS of $music$, $sound$, and $background$ image played on a state s of a *karaoke* object. *Q*(s) indicates QoS of the state s of the *karaoke* object. *Q*(s_1) *dominates* *Q*(s_2) (*Q*(s_1) $\succeq$ *Q*(s_2)) iff a state s_1 supports better QoS than another state s_2. For example, $\langle$30[fps], 16[colours]$\rangle$ $\preceq$ $\langle$40, 64$\rangle$. The formal definition of the relation $\preceq$ is discussed in papers [9, 10] Since "$\preceq$" is a partially ordered relation, a *least upper bound* (lub) $q_1 \cup q_2$ of QoS q_1 and q_2 is some QoS q_3 in S such that 1) $q_1 \preceq q_3$ and $q_2 \preceq q_3$, and 2) no QoS q_4 where $q_1 \preceq q_4 \preceq q_3$ and $q_2 \preceq q_4 \preceq q_3$. For example, $\langle$30[fps], 1024[colours]$\rangle$ $\cup$ $\langle$40[fps], 512[colours]$\rangle$ = $\langle$40, 1024$\rangle$.

An application requires an object to support *requirement* QoS (*RoS*). Let *r* be RoS of an application. If *Q*($play$(s)) $\succeq$ *r*, the application can accept the *karaoke* service supported thought the method $play$ since enough QoS is supported.

An object k_1 created from the *karaoke* class is also composed of a $music$ object m_1, $words$ object w_1, and *background* object b_1. Another *karaoke* object k_2 is same as k_1 except that the *background* object of k_2 is b_2 ($\neq b_1$). An application considers a pair of the objects k_1 and k_2 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, the component class is referred to as *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*, respectively. In addition, if *c* is not significant, *c* is referred to as *N* significant. *stype*(c) shows a significant type of a class *c* ($\in$\{ *SQ*, *S*, *Q*, *N* \}). For example, classes $words$ and $music$ are *SQ*-types. Because no people can do the *karaoke* without both the classes with enough QoS. On the other hand, a class *background* is an *N*-type, in which applications are not interested. Mandatory and optional classes [13, 14] show *SQ* and *N*-types, respectively. We extend the consistent relations by newly considering the significant types *SQ*, *S*, *Q*, and *N* of component classes. There are following types of consistent relations between a pair of states s_t and s_u of a class c. Here, let c_i indicate a component class of a class c. For a pair of significant types α and β, "α / β" stands for "α or β".

- s_t is $state$ and *QoS* (*SQ*) *consistent* with s_u ($s_t - s_u$) iff s_t = s_u.
- s_t is *state* (*S*) *consistent* with s_u ($s_t \sim s_u$) iff s_t and s_u are obtained by degrading QoS of some state s of c.
- s_t is *QoS* (*Q*) *consistent* with s_u ($s_t \frown s_u$) iff $s_t \sim s_u$ and *Q*(s_t) = *Q*(s_u).
- s_t is *semantically SQ* (*Sem-SQ*) *consistent* with s_u ($s_t \equiv s_u$) iff $s_t - s_u$ or $c_i(s_t) \equiv c_i(s_u)$ for every *SQ*-type component class c_i.
- s_t is *semantically S* (*Sem-S*) *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*/*S*-type component class c_i.
- s_t is *semantically Q* (*Sem-Q*) *consistent* with s_u ($s_t \frown s_u$) iff $s_t \frown s_u$ or $c_i(s_t) \frown c_i(s_u)$ for every *Q*-type component class c_i.

Furthermore, there are following types of consistent relations with respect to *RoS*. Here, let *r* be some *RoS* instance.

- s_t is *state* and *RoS r* (*S*[*r*]) *consistent* with s_u($s_t -_{[r]} s_u$) iff $s_t - s_u$ and $Q(s_t) \cap Q(s_u) \succeq$ *r*.
- s_t is [*r*] *consistent* with s_u ($s_t \frown_{[r]} s_u$) iff $s_t \frown s_u$ and $Q(s_t) \cap Q(s_u) \succeq$ *r*.
- s_t is *semantically S*[*r*] (*Sem-S*[*r*]) *consistent* with s_u ($s_t \equiv_{[r]} s_u$) iff $s_t -_{[r]} s_u$ or $c_i(s_t) \equiv_{[r]} c_i(s_u)$ for every *Q*-type component class c_i.
- s_t is *semantically* [*r*] (*Sem*-[*r*]) *consistent* with s_u ($s_t \frown_{[r]} s_u$) iff $s_t \frown_{[r]} s_u$ or $c_i(s_t) \frown_{[r]} c_i(s_u)$ for every *Q*-type component class c_i.

If a pair of states s_t and s_u are SQ-consistent with one another, (s_t - s_u), both state and QoS of s_t and s_u are the same, i.e. $s_t = s_u$ and *Q*(s_t) = *Q*(s_u). In Figure 1, suppose every image object is fully colored and *sound* object supports a stereo type of sound. A state s_1 is obtained by changing state *s* of the *car* object with *monochromatic* image. Another state s_2 is obtained by changing the *sound* object with *monochromatic* one. Here, s_1 and s_2 are S-consistent ($s_t \frown s_u$) but not SQ-consistent (s_t s_u). Thus, s_t is S-consistent with s_u but *Q*(s_t) and *Q*(s_u) may not be the same.

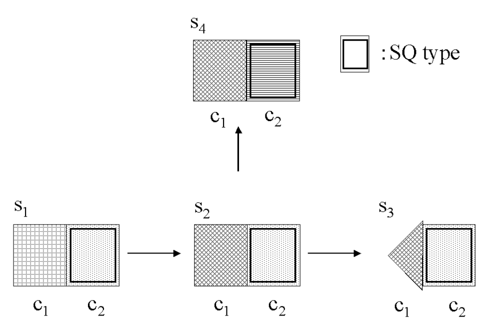

Figure 3. State and QoS change.

Figure 3 shows state and QoS change of an object of a class c with two component classes c_1 and c_2. First, QoS of a state s_1 of the component class c_1 of

c is changed to another state s_2 and then the state of c_1 is changed to s_3. Here, suppose c_1 and c_2 are N- and SQ-types, respectively. A pair of the states s_1 and s_2 are S-consistent ($s_1 \sim s_2$) because $c_2(s_1) = c_2(s_2)$ and $c_1(s_2)$ is obtained just by changing QoS of $c_1(s_1)$. In addition, the states s_1, s_2, and s_3 are semantically SQ-consistent ($s_1 \equiv s_2 \equiv s_3$) since $c_2(s_1) = c_2(s_2) = c_2(s_3)$ for the SQ-type class c_2. Next, suppose QoS of the component class c_2 of s_2 is changed to s_4. Here, s_2 and s_4 are not SQ-consistent because $c_2(s_2) \neq c_2(s_4)$. Let *R* be a set of possible RoS instances, named *RoS set*. Let $-_R$, $\equiv_R$, $\frown_R$ and $\stackrel{\frown}{=}_R$ show sets $\{ -_{[r]} \mid r \in r \}$, $\{ \equiv_{[r]} \mid r \in r \}$, $\{ \frown_{[r]} \mid r \in r \}$, and $\{ \stackrel{\frown}{=}_{[r]} \mid r \in r \}$ which are referred to as SR-consistent, *semantically* SR-consistent, *R*-consistent, and *semantically* R-consistent relations, respectively. Here, let *SQ*, *S*, *Q*, *Sem-SQ*, *Sem-S*, *Sem-Q*, *SR*, *R*, *Sem-SR*, and *Sem-R* denote *consistency* sets of possible *SQ*, *S*, *Q*, *sem-SQ*, *Sem-S*, *Sem-Q*, *SR*, *R*, *Sem-SR*, and *Sem-R* consistent relations of a class c. Let **C** be a *consistency family* which is a family {*SQ*, *S*, *Q*, *Sem-SQ*, *Sem-S*, *Sem-Q*, *SR*, *R*, *Sem-SR*, *Sem-R*} of the sets of the consistent relations. For a consistency set α in the consistency family **C**, let $\Box_\alpha$ show an α-consistent relation. For example, $\Box_\equiv$ stands for the semantically *SQ* (*Sem-SQ*) consistent relation $\equiv$. For a pair of methods op_1 and op_2 of a class c, "$op_1 \ \Box_\alpha \ op_2$" shows that an α-consistent relation $op_1(s) \ \Box_\alpha \ op_2(s)$" holds for every state s of a class c. For a pair of consistency sets α and β in **C**, "α *dominates* β" ($\alpha \rightarrow \beta$) means $\alpha \subseteq \beta$, showing that $s_t \ \Box_\beta \ s_u$ if $s_t \ \Box_\alpha \ s_u$ for every pair of states s_t and s_u of a class c. Figure 4 shows a Hasse diagram where a node α shows a consistency set α in **C** and a directed edge from a node α to another node β shows a dominant relation "$\alpha \rightarrow \beta$". For example, "$SQ \rightarrow$ *Sem-SQ*" means that $s_1 \equiv s_2$ if $s_1 - s_2$ for every pair of states s_1 and s_2. Let r_1 and r_2 show a pair of RoS instances, i.e. $r_1, r_2 \in R$. That is, r_1 dominates r_2 ($r_1 \rightarrow r_2$) iff $r_1 \succeq r_2$. Here, r_1 = ⟨40[fps], 1024[colours]⟩ $\succeq r_2$ = ⟨30[fps], 512[colours]⟩. Suppose $r_1 \rightarrow r_2$. $s_1 \equiv_{[r2]} s_2$ if $s_2 \equiv_{[r1]} s_1$. $[r_1] \leftarrow [r_2]$, $S[r_1] \leftarrow S[r_2]$, *Sem-S*$[r_1] \leftarrow$ *Sem-S*$[r_2]$, and $Sem[r_1] \leftarrow Sem[r_2]$.

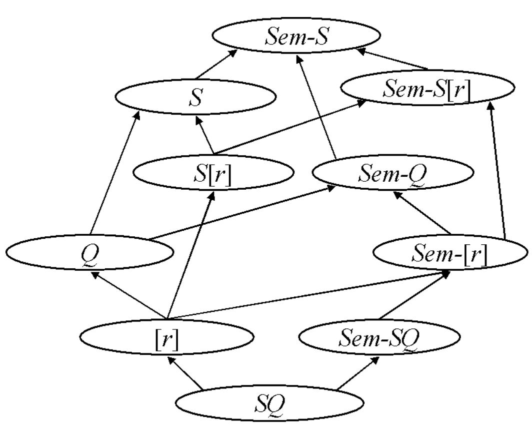

Figure 4. Hasse diagram.

Now, we define an α-compatible relation $\Diamond_\alpha$ on methods of a class c for α in **C**.

[**Definition**] A method op_t is α-*compatible* with a method op_u for a consistency set α in **C** ($op_t \ \Diamond_\alpha \ op_u$) iff $op_t \circ op_u(s) \ \Box_\alpha \ op_u \circ op_t(s)$ for every state s of a class c. □

op_t is referred to as α-*conflict* with op_u ($op_t \ \not\Diamond_\alpha \ op_u$) iff op_t is not α-compatible with op_u. Here, $\Diamond_\alpha$ is assumed to be symmetric but not transitive. Let α and β be consistency sets in **C**. If $op_t \ \Diamond_\alpha \ op_u$, op_t and op_u are allowed to be performed in any order with respect to a consistency set α.

[**Theorem**] $\Diamond_\alpha \subseteq \Diamond_\beta$ iff $\alpha \leftarrow \beta$. □

A same Hasse diagram as Figure 4 holds for conflicting and compatible relations. For example, $op_t \ \Diamond_\equiv \ op_u$ if $op_t \ \Diamond_- \ op_u$ since *Sem-SQ* $\leftarrow$ *SQ*.

Next, let us consider whether or not a pair of methods op_t and op_u can be concurrently performed with respect to some consistency set α in **C**. Let $\{ op_t \parallel op_u(s) \}$ be a set of possible states obtained by concurrently performing op_t and op_u on a state s of a class c.

[**Definition**] A pair of methods op_t and op_u are α-*independent* iff for every state s of a class c, s' $\Box_\alpha \ op_1 \circ op_2(s)$ or s' $\Box_\alpha \ op_2 \circ op_2(s)$ for every state s' $\in \{ op_1 \parallel op_2(s) \}$. □

Here, op_t and op_u are α-*exclusive* iff op_t and op_u are not α-*independent*. If op_t and op_u are α-*exclusive*, op_t and op_u cannot be concurrently performed on a state of a class c with respect to the consistency set α. If op_t and op_u are concurrently performed, a state obtained by $op_t \parallel op_u$ is not consistent with respect to α. The α-*exclusive* relations are also represented in a same Hasse diagram as Figure 4.

3 Hybrid Concurrency Control

3.1 Timestamp ordering(TO) scheduler

Each object is provided with two types of concurrency control mechanisms; a timestamp ordering (TO) scheduler [1] and locking protocol [1] [Figure 5]. The TO scheduler is used to serialize conflicting methods issued by transactions with consistency sets. An object is locked in order to realize mutual exclusion among methods. Each transaction T is assigned a timestamp $ts(T)$ which shows a local time when T is initiated. For every pair of different transactions T_1 and T_2, either $ts(T_1) < ts(T_2)$ or $ts(T_1) > ts(T_2)$. 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.

Each transaction T manipulates objects according to some consistency set α in the consistency family **C** ($type(T) = \alpha$). T issues a method op, where $type(op) = type(T)$. Transactions issue requests of methods to the TO scheduler of an object o. The methods are buffered and are ordered in the TO scheduler according to the following timestamp ordering (TO) rule:

[**Consistent timestamp ordering (TO) rule**] For every pair of methods op_1 and op_2 on an object o, op_1 $precedes$ op_2 in the TO scheduler of the object o ($op_1 \Rightarrow_o op_2$) if op_1 α-conflicts with op_2 ($op_1 \ \not\Diamond_\alpha \ op_2$), $ts(op_1) < ts(op_2)$, and $\alpha = type(op_1) \cap type(op_2)$. □

Suppose there are a pair of transactions T_1 and T_2 where $ts(T_1) < ts(T_2)$. The transaction T_1 issues a method *grayscale* and the other transaction T_2 issues another method *add-car* to the *background* object b of Figure 1. Suppose $type(T_1) = type(T_2) = Q$. Since *grayscale* Q-conflicts with *add-car*, T_1 Q-conflicts with T_2. *grayscale* has to precede *add-car* in the TO scheduler of the object b (*grayscale* $\Rightarrow_b$ *add-car*) since ts (*grayscale*) $< ts$ (*add-car*). Next, suppose $type(T_1)$ = Q and $type(T_2)$ = r where RoS [r] shows "application is not interested in the colour of a car." $Q \cap [r]$ = [r] since *grayscale* and *add-car* are [r]-*compatible* (*grayscale* $\Box_{[r]}$ *add*). Hence, *add-car* $\Rightarrow_b$ *grayscale* even if ts(*grayscale*) $< ts$(*add-car*).

First, suppose a transaction T issues a method op to the TO scheduler of an object o. There is a variable $mts(op)$ showing the timestamp of a method op which is most recently started on the object o. The variable $mts(op)$ is initially 0. The method op is stored in the TO scheduler according to the ordering rule as "op' precedes op ($op' \Rightarrow_o op$)" if $mts(op') < ts(op)$ for every method op' α-conflicting with the method op. Otherwise, op is rejected and then the transaction T is aborted. The number of transactions to be aborted can be reduced if a top method op of the TO scheduler is delayed as discussed in the paper [1].

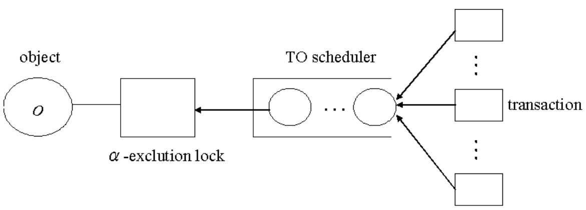

Figure 5. TO scheduler and locking.

Suppose there are a pair of transactions T_1 and T_2 which manipulate a *background* object b where $ts(T_1) < ts(T_2)$. The transaction T_1 issues a method *grayscale* and T_2 issues *add-car* to the object b. Suppose $type(T_1)$ = Q and $type(T_2)$ = S. $S = Q \cap S$. Since *grayscale* S-conflicts with *add-car*, *grayscale* is required to precede *add-car* in the TO-scheduler of the object b. Next, suppose $type(T_1) = type(T_2) = r'$ ("not interested in colour of car"). Since *grayscale* is [r']-compatible with *add-car*, *add-car* can precede *grayscale* in the TO scheduler of the object b

3.2 Serializability

Let T_i and T_j be a pair of transactions issuing methods op_i and op_j to an object o, respectively. The transaction T_i α-*precedes* T_j with respect to a consistency set α ($T_i \xrightarrow{\alpha} T_j$) iff op_i α-*conflicts* with op_j ($op_i \not\Diamond_\alpha op_j$) where $\alpha = type(T_i) \cap type(T_j)$ and op_i is started before op_j on the object o. The α-precedent relation $\xrightarrow{\alpha}$ is transitive.

[**Theorem**] A transaction T_i α'-precedes another transaction T_j ($T_i \xrightarrow{\alpha'} T_j$) with respect to a consistency set α' if $T_i \xrightarrow{\alpha} T_j$ and $\alpha' \rightarrow \alpha$. $\Box$

[α-**Serializability**] A collection ***T*** of transactions $T_1, \cdots, T_m$ are α-$serializable$ with respect to a consistency set α if both $T_i \xrightarrow{\alpha} T_j$ and $T_j \xrightarrow{\alpha} T_i$ do not hold for every pair of transactions T_i and T_j where $\alpha = type(T_i) \cap type(T_j)$. $\Box$

Let SQ, S, Q, R, Sem-SQ, Sem-S, Sem-Q, and Sem-R be sets of possible transactions which are SQ, S, Q, R, Sem-SQ, Sem-S, Sem-Q, and Sem-R-serializable, respectively. Let **SR** be a family $\{SQ$, S, Q, R, Sem-SQ, Sem-S, Sem-Q, and Sem-$R\}$ of the transaction sets. For a pair of transaction sets α_1 and α_2 in **SR**, "$\alpha_1 \rightarrow \alpha_2$" shows $\alpha_1 \subseteq \alpha_2$. Let ***T*** be a set of $\{T_1, \ldots, T_n\}$ of transactions. Suppose $\alpha_1 \rightarrow \alpha_2$ for α_1, $\alpha_2 \in$ *SR*. ***T*** is α_2-serializable if ***T*** is α_1-serializable. For example, ***T*** is Q-serializable if ***T*** is S-serializable. The Hasse diagram for **SR** and $\rightarrow$ is isomorphic with Figure 4.

[**Serializability**] A set ***T*** of transactions is α-*serializable* iff $\xrightarrow{\alpha}$ is acyclic for every consistent set α in **C**. $\Box$

3.3 Locking protocol

A top method op in the TO scheduler is first taken. We have to decide whether or not the method op can be performed on an object o. A set of methods which are being performed on the object o is stored in a variable R_o. If the method op satisfies the following execution rule, op is removed from the TO scheduler and is performed on the object o:

[**Execution rule**] If one of the following rules is satisfied, a method op is performed on an object o,

1. R_o is empty.
2. If R_o is not empty, op is not α-exclusive with every method op' in R_o where $\alpha = type(op) \cap \{type(op') \mid op' \in R_o\}$. $\Box$

If the method op completes, op is removed from R_o. If op does not satisfy the execution rule, the method op waits in the TO scheduler.

In order to realize the execution rule, the locking mechanism is adopted. For a top method op in the TO scheduler, a lock request of a mode $\mu_\alpha(op)$ is issued to the object o where $\alpha = type(op)$. For every method op' in R_o, if $\mu_\alpha(op')$ is not α''-exclusive with μ_α are $\alpha'' = \alpha' \cap \alpha$, the object o is locked in the mode $\mu_\alpha(op)$. If succeeded in locking the object o, the method op is started performed and op is added to R_o. Here, $mts(op)$:= max($ts(op)$, $mts(op)$). Otherwise, the method op is kept waited in the TO scheduler.

Suppose the top method op in the TO scheduler does not satisfy the execution rule. The method op has to stay in the TO scheduler. Until the top method satisfies the execution rule, i.e. the object is locked, every method has to wait in the TO scheduler. In order to increase the throughput, another methods than the top method is tried to be performed. A method which is α-compatible with op and preceded by op in the TO scheduler can be performed on the object o.

[**Definition**] A method op is α-$ready$ in the TO scheduler of an object o with respect to a consistency set α iff op satisfies the α-execution rule and every method op' preceding op in the TO scheduler is α-compatible with op and $\alpha = type(op) \cap type(op')$. $\Box$

An α-ready method op is referred to as top α-$ready$ method in the TO scheduler iff op precedes every other α-$ready$ methods in the TO scheduler. If the top α-$ready$ method op satisfies the execution rule, op is removed from the TO scheduler and then is performed on the object o. This is repeated until there is no α-$ready$ method in the TO scheduler.

Suppose there are a pair of transactions T_1 and T_2 where $ts(T_1) < ts(T_2)$. T_1 issues a pair of methods op_1 and op_2 to objects x and y, respectively. T_2 issues op_2 and op_4 to x and y, respectively. S_x and S_y show the TO schedulers of objects x and y, respec-

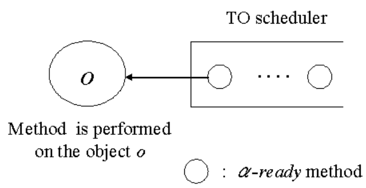

Figure 6. α-*ready* **method.**

tively. Suppose that op_1 $Sem\text{-}SQ$- conflicts with op_2 ($op_1 \not\Diamond_{\equiv} op_2$) and ($op_3 \not\Diamond_{\equiv} op_4$). op_1 precedes op_2 in the TO scheduler S_x ($op_1 \Rightarrow_x op_2$) and $op_3 \Rightarrow_y op_4$. In the TO scheduler S_x, op_1 is removed and is performed on the object x. Then, op_2 is examined for the execution rule. Since op_2 is $Sem\text{-}SQ$-exclusive with op_1, op_1 is kept waited in S_x until op_1 completes. On the other hand, after op_3 is started on the object y, op_4 is performed because op_4 is not $Sem\text{-}SQ$-exclusive with op_3.

If a method op completes and the lock of op is released, the procedure presented here is applied from the top method in the TO scheduler.

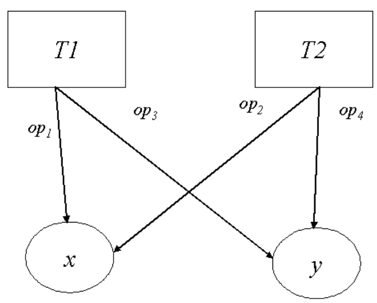

Figure 7. Concurrent access.

3.4 Commitment

We discuss how a transaction terminates. A transaction issues special types of methods $commit(c)$ and $abort(a)$ to objects in addition to methods to manipulate the objects. Suppose a transaction T issues methods $op_1, \cdots, op_m$ to an object o. After the methods $op_1, \cdots, op_m$ are performed on the object o, the transaction T issues a commit method c to the object o, and the commit method c is performed on the object o. Here, the locks held by the methods $op_1, \cdots, op_m$ are released. Similarly, the locks are released if $abort(a)$ is performed. That is, a strict two-phase locking protocol [1] is adopted. Each of commit and abort methods is timestamped as well as the other methods.

[**Definition**] Let e_1 and e_2 be commit or abort methods of an object o issued by transactions T_1 and T_2, respectively. e_1 precedes e_2 in the TO scheduler ($e_1 \Rightarrow_o e_2$) if $ts(e_1) < ts(e_2)$ and T_1 α-conflicts with T_2 when $\alpha = type(T_1) \cap type(T_2)$. □

Suppose a top method is a commit method c of a transaction T in the TO scheduler of an object o. If the commit c satisfies the execution rule, c is removed. The object o is physically updated and the lock of the object o is released. If the top method is an abort method a, the lock of the object o is just released.

4 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. Based a consistent relation, we defined the conflicting relations. We discussed concurrently control with two mechanisms time ordering (TO) and locking protocols.

References

[1] P. A. Bernstain, V. Hadzilaces, and G. N. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[2] A. Cambell, G. Coulson, F. Garcfa, D. Hutchison, and H. Leopold. Integrated Quality of Service for Multimedia Communication. *Proc. of IEEE InfoCom*, pages 732–793, 1993.

[3] A. Campbell, G. Coulson, and D. Hutchison. A Quality of Service Architecture. *ACM SIGCOMM Comp. Comm. Review*, 24:6–27, 1994.

[4] D. Gall. Mpeg: A Video Compression Standard for Multimedia Applications. *Comm. ACM*, 34(4):46–58, 1991.

[5] J. Garza and W. Kim. Transaction Management in an Object-Oriented Database System. *Proc. SIGMOD Conf.on Management of Data,*, 1989.

[6] J. Gray. Notes on Database Operating Systems. *Lecture Notes in Computer Science*, (60):393–481, 1978.

[7] J. Grosling and H. McGilton. *The Java Language Environment*. Sun Microsystems Inc., 1996.

[8] O. M. G. Inc. *The Common Object Request Broker: Architecture and Specification, Rev2.0*. 1995.

[9] T. Kanezuka and M. Takizawa. Qos Oriented Flexibility in Distributed Objects. *Proc. of Int'l Symp. on Comm.(ISCOM'97)*, pages 144–148, 1997.

[10] T. Kanezuka and M. Takizawa. Quality-based Flexibility in Distributed Objects. *Proc. of 1st IEEE Int'l Symp. on Object-oriented Real-time Distributed Computing (ISORC'98)*, pages 350–357, 1998.

[11] H. F. Korth, E. Levy, and A. Silberschalz. A Formal Approach to Recovery by Compensating Transactions. *Proc. of VLDB*, pages 95–106, 1990.

[12] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *CACM*, 21(7):558–565, 1978.

[13] N. Nemoto, K. Tanaka, and M. Takizawa. Quality-Based Approach to Locking Multimedia Objects. *Journal of Internet Technology, special issue on "Internet Multimedia Information Systems"*, 2(4):309–316, 2001.

[14] N. Nemoto, K. Tanaka, and M. Takizawa. Quality-Based Concurrency Control for Multimedia Objects. *Proc of the 7th International Conference on Distributed Multimedia Systems (DMS'2001)*, pages 218–225, 2001.

[15] C. B. Owen and F. Makedon. *Computed Synchronization for Multimedia Applications*. Kluwer Academic, 1999.

[16] Z. Tari and O. Bukhres. *Fundamentals of Distributed Object Systems*. Wiley, 2001.

[17] Z. Wang. *Internet QoS*. Morgan Kaufmann, 2001.

[18] M. Yokoyama, N. Nemoto, K. Tanaka, and M. Takizawa. Quality-Based Approach to Manipulating Multimedia Objects. *Proc. of 2000 International Conf. on Information Society in the 21 Century: Emerging Technologies and New Challenges (IS2000)*, pages 380–387, 2000.