マルチメディア通信と分散処理 99－3
（2000．9．22）

Quality-Based Approach to Locking Multimedia Objects

Naokazu Nemoto, Katsuya Tanaka, and Makoto Takizawa
Tokyo Denki University
Email {nemoto, katsu, taki}@takilab.k.dendai.ac.jp

It is critical for applications to obtain enough quality of service (QoS) from multimedia objects. Not only states but also QoS of objects are changed by performing methods on the objects. Each object is required to be consistent in presence of multiple transactions issuing requests to the objects. We discuss new types of equivalent and conflicting relations among methods with respect to QoS. We introduce two types of locking modes, serialization and mutually exclusive modes to synchronize concurrent accesses to objects based on the QoS relations. We also discuss how those lock modes conflict.

QoS に基づくマルチメディアオブジェクトの同期手法

根本 直一，田中 勝也，滝沢 誠
東京電機大学
E-mail {nemoto, katsu, taki}@takilab.k.dendai.ac.jp

分散アプリケーションでは複数のマルチメディアオブジェクトが操作される。マルチメディアオブジェクトは、メソッドを介してのみ操作される。メソッドの実行により、オブジェクトの状態とともにオブジェクトの提供する QoS も変化する。本論文では、新たに QoS を考慮した演算間の競合関係を定義する。また、オブジェクトをロックするモードとして、直列可能型と排他制御型の 2 つを提案する。QoS に基づいて、必要なメソッドの実行を順序付けすることにより、システムのスループット向上をはかる。

1 Introduction

In distributed applications, service supported by each multimedia object is characterized by parameters showing *quality of service* (*QoS*) like frame rate. Not only the state but also QoS of the object are changed by performing the methods. For example, suppose a *movie* object suppose to a pair of methods, *adds* and *grayscale*, which *add* and *grayscale* changes to *monochromatic* version, respectively. If *grayscale* is performed after a colored *car* is *added*, the *movie* is monochromatic. On the other hand, if the car is *added* after *grayscale*, only the car is colored but the others are monochromatic in the *moive*. If the application is not interested in the color, both the *movies* are equivalent. Thus, *add* and *grayscale* can be considered not to conflict from the QoS point of view. We discuss new types of conflicting relations based a the QoS concept in this paper.

Multiple methods are issued by multiple transactions. Here, the objects are kept to be consistent. According to the synchronization theories [1], a pair of conflicting methods on an object are serially performed by locking the object. A pair of compatible methods can be performed in any order. In addition, some compatible methods can be concurrently performed. However, a pair of compatible methods *increment* and *decrement* of a *counter* object cannot be concurrently performed. Furthermore, even some pair of conflicting methods can be concurrently performed while it is critical to decode which methods to be started before the other. We newly introduce two orthogonal types of lock modes, *serialization* and *mutually exclusive modes* based on QoS. The serialization locks [1] are used to serialize the computation of conflicting methods while the mutually exclusive locks are used to mutually exclusively perform methods. In this paper, we discuss how these lock modes are related with respect to QoS.

In section 2, we present a system model. In section 3, we discuss conflicting relations among methods based with respect to QoS. In section 4, we discuss how to lock objects to keep the objects consistent with respect to QoS.

2 System Model

A system is composed of multiple objects. An object is an encapsulation of data and methods for manipulating the data. There are two types of objects, *classes* and *instances*. A class c is composed of *attributes* $A_1, \ldots, A_m$ $(m \geq 0)$ and *methods* $op_1, \ldots, op_l$ $(l \geq 1)$. An instance o is created from the class c. A tuple $\langle v_1, \ldots, v_m \rangle$ of values is a *state* of the instance o where each v_i is a value taken by A_i Let *objects* show *instances* in this paper as used in Java [6] and C++ [13]. Each object has one state at a time. A *state* of a class means a state of an object of the class.

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 $c_i(s)$ denote a projection of a state s of the class c to a component class c_i. For example, a class *Karaoke* is composed of three component classes, *music*, *words*, and *background*. *music*(k) shows a state of *music* in a *Karaoke* object k. The *background* class is furthermore composed of *car*, *tree*, and *cloud* classes.

On receipt of a request of a method *op*, *op* is performed on an object o. Let *op*(s) denote a state obtained by performing *op* on a state s of o, re-

spectively. Here, $op_1 \circ op_2$ and $op_1 \parallel op_2$ show that a pair of methods op_1 and op_2 are serially and concurrently performed, respectively.

Applications obtain service from a multiple object only through methods of the object. Each service is characterized by *quality of service* (QoS) like level of resolution. Each state s of an object o supports a QoS instance denoted by $Q(s)$. $Q(s_1)$ *dominates* $Q(s_2)$ ($Q(s_1) \succeq Q(s_2)$) iff s_1 supports better QoS than s_2. The formal definition of the dominant relation $\preceq$ is discussed in the paper[7,8]. Since $\preceq$ is a partially ordered relation, a *least upper bound* (*lub*) $q_1 \cup q_2$ of QoS instances q_1 and q_2 are some QoS instance q_3 in S such that 1) $q_1 \preceq q_3$ and $q_2 \preceq q_3$, and 2) there is no instance q_4 in S where $q_1 \preceq q_4 \preceq q_3$ and $q_2 \preceq q_4 \preceq q_3$.

An application requires an object to support some QoS which is referred to as *requirement* QoS (*RoS*). Let r be an RoS instance.

3 QoS Based Conflicting Relations

3.1 Equivalent relations

A class *movie* is composed of two component classes; *advertisement* and *content*. An object m_1 created from the *movie* class is also composed of an *advertisement* object a_1 and a *content* object c_1. Another *movie* object m_2 is a same as m_1 except that the *advertisement* object of m_2 is a_2. An application does not care the difference between m_1 and m_2 since the application is interested in only the content of *movie*. Here, m_2 is considered to be *equivalent* with m_1. A class like *content* in which applications are interested is *mandatory*. On the other hand, a class like *advertisement* is *optional*. In addition, m_1 and m_2 support the same level of QoS, i.e. $Q(m_1) = Q(m_2)$. Suppose a class c is composed of classes $c_1, \ldots, c_m$ ($m \geq 0$). An application specifies whether each c_i is *mandatory* or *optional*. Every object o of c is required to include an object o_i of a mandatory class c_i. If c_i is optional, o may not include any object of c_i.

There are the following equivalent relations between a pair of states s_t and s_u of a class c:

- s_t is *state-equivalent* with s_u ($s_t - s_u$) iff $s_t = s_u$.
- s_t is *semantically equivalent* 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 mandatory component class c_i of c.
- s_t is *QoS-equivalent* with s_u ($s_t \approx s_u$) iff $s_t - s_u$ or s_t and s_u are obtained by degrading QoS of some state s of c.
- s_t is *semantically QoS-equivalent* with s_u ($s_t \cong s_u$) iff $s_t \approx s_u$ or $c(s_t) \cong c(s_u)$ for every mandatory component class c_i of c.
- s_t is *RoS-equivalent* with s_u on RoS R ($s_t -_R s_u$) iff $s_t \approx s_u$ and $Q(s_t) \cap Q(s_u) \succeq R$.
- s_t is *semantically RoS-equivalent* with s_u on RoS R ($s_t \equiv_R s_u$) iff $s_t -_R s_u$ or $c_i(s_t) \equiv_R c_i(s_u)$ for every mandatory class c_i of c.

[**Example 1**] Let K be an object created from the *Karaoke* class [Figure 1], which supports three methods *sound1*, *sound2*, and *sound3* which derive states s_1, s_2, and s_3 from a state s of K, respectively [Figure 2]. Stereo sound of *music* m_1 is played while a *background* object *bg1* with a *words* object w_1 is displayed in the state s_1. *sound2* plays stereo sound of *music* m_2 while displaying the *words* object w_2 and a *background* object *bg2* which includes a *car* object. Suppose an application is interested only in *music* and *words*. Here, *music* and *words* are mandatory but *background* is optional in *Karaoke*. $s_1 \equiv s_2$ while $s_1 \neq s_2$ and $Q(s_1) = Q(s_2)$. On the other hand, *sound3* outputs a state s_3 of the *Karaoke* object K, which shows only a monaural sound of *music* m_3 and *words* without *background*. Here, $s_1 \not\equiv s_3$ because $Q(s_1) \neq Q(s_3)$. $m_1 \approx m_3$. *sound1* supports a higher level of QoS than *sound3*. Suppose RoS R is that the application is not interested in the quality of sound. Here, $m_1 -_R m_2$. Furthermore, if *music* and *words* are mandatory, $s_1 \equiv_R s_3$ since $m_1 -_R m_3$. □

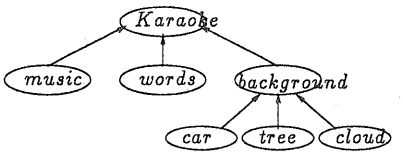

Figure 1: Karaoke class.

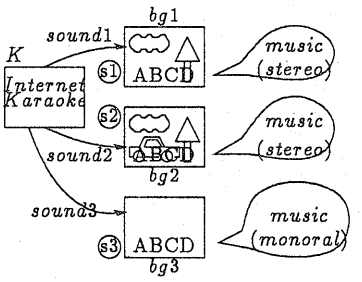

Figure 2: Karaoke object.

There are the following relations between a pair of methods op_t and op_u of a class c :

- op_t is *state-equivalent* with op_u ($op_t - op_u$) iff $op_t(s) - op_u(s)$ for every state s of c.
- op_t is *semantically equivalent* with op_u ($op_t \equiv op_u$) iff $op_t(s) \equiv op_u(s)$ for every state s of c.
- op_t is *QoS-equivalent* with op_u ($op_t \approx op_u$) iff $op_t(s) \approx op_u(s)$ for every state s of c.
- op_t is *semantically QoS-equivalent* with op_u ($op_t \cong op_u$) iff $op_t(s) \cong op_u(s)$ for every state s of c.
- op_t is *RoS-equivalent* with op_u on RoS R ($op_t -_R op_u$) iff $op_t(s) -_R op_u(s)$ for every state s of c.

- op_t is *semantically RoS-equivalent* with op_u on R ($op_t \equiv_R op_u$) iff $op_t(s) \equiv_R op_u(s)$ for every state s of c.

Let *State*, *Sem*, *QoS*, *RoS*, *Sem-QoS*, and *Sem-RoS* be sets of possible state-, semantically, QoS-, RoS-, semantically QoS-, and semantically RoS-equivalent relations of a class c. Let E be a family of *State*, *Sem*, *QoS*, *RoS*, *Sem-QoS*, and *Sem-RoS*. For a pair of sets α and β in E, "$\alpha \rightarrow \beta$" means $\alpha \subseteq \beta$, showing that every pair of operations op_1 and op_2 satisfy the β-equivalency if op_1 and op_2 satisfy the α-equivalency. Figure 3 shows a Hasse diagram where a node α shows a set in E and a directed edge fomr α to β shows "$\alpha \rightarrow \beta$" . For example "*State* $\rightarrow$ *Sem*" means that $op_1 \equiv op_2$ if $op_1 - op_2$ for any pair of methods op_1 and op_2. $op_1 \cong op_2$ if $op_1 \equiv op_2$. $op_1 \cong op_2$ if $op_1 \approx op_2$.

[**Theorem**] The Hasse diagram as shown in Figure 3 holds for the α-equivalent relations. □

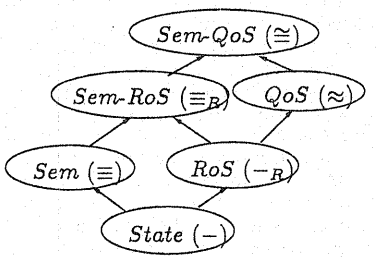

Figure 3: Equivalent relations.

3.2 Compatible relations

In the traditional theories [1, 9], a conflicting relation among methods is defined on the basis of states of an object. A method op_t *conflicts* with another method op_u in an object o iff the result obtained by performing op_t and op_u depends on the computation order. Multimedia objects are characterized by QoS in addition to the states. There are the following compatible relations between a pair of methods op_t and op_u of a class c:

- op_t is *state-compatible* with op_u ($op_t \mid op_u$) iff $op_t \circ op_u - op_u \circ op_t$.
- op_t is *QoS-compatible* with op_u ($op_t \parallel op_u$) iff $op_t \circ op_u \approx op_u \circ op_t$.
- op_t is *RoS-compatible* with op_u on RoS R ($op_t \mid_R op_u$) iff $op_t \circ op_u -_R op_u \circ op_t$.
- op_t is *semantically compatible* with op_u ($op_t \mid\mid\mid op_u$) iff $op_t \circ op_u \equiv op_u \circ op_t$.
- op_t is *semantically QoS-compatible* with op_u ($op_t \between op_u$) iff $op_t \circ op_u \cong op_u \circ op_t$.
- op_t is *semantically RoS-compatible* with op_u on R ($op_t \mid\mid\mid_R op_u$) iff $op_t \circ op_u \equiv_R op_u \circ op_t$.

Let "α-*compatible* relation ($\diamond_\alpha$)" show some of the compatible relations where $\alpha \in \{$*State*, *Sem* (*semantically*), *QoS*, *RoS*, *Sem-QoS* (*semantically-QoS*), *Sem-RoS* (*semantically-RoS*)$\}$. For example, $\diamond_{QoS}$ shows $\parallel$ and $\diamond_R$ shows $\mid_R$. op_t α-*conflicts* with op_u ($op_t \not\diamond_\alpha op_u$) unless $op_t \diamond_\alpha op_u$. For example, op_t QoS-conflicts with op_u ($op_t \not\parallel op_u$) unless $op_t \parallel op_u$. $\diamond_\alpha$ is symmetric but is not transitive.

[**Example 2**] Suppose the *movie* object m is composed of a *background* object *bg* and a *car* object c, which supports methods *add*, *grayscale*, *mediascale*, and *reduce*. *add* adds a *car* object in the *movie*. *grayscale* degrades a colored video to a white-black gradation video. *mediascale* reduces a frame rate to half of the original one. *reduce* decreases a number of colors to 16 colors. Suppose an application obtains an object m_1 by performing *grayscale* on the object m after *add*, i.e. $m_1 = add \circ grayscale(m)$. Here, m_1 is a white-black gradation video with the *background* object composed of the car. On the other hand, an object m_2 shows a colored *car* and the white-black background. Here, $Q(m_2) \neq Q(m_1)$. An object obtained by *add* $\circ$ *grayscale* supports a different level of QoS from *grayscale* $\circ$ *add*. *add* QoS-conflicts with *grayscale* (*add* $\not\parallel$ *grayscale*). By performing *grayscale* $\circ$ *add*, the response data of *add* shows white-black gradation *background* and colored *car*. However, the white-black gradation video is obtained by *add* $\circ$ *grayscale*. If the application is only interested in a colored *car*, the response data obtained by *grayscale* $\circ$ *add* satisfies the application requirement R. However, a response data obtained by *add* $\circ$ *grayscale* does not satisfy R. That is, *add* $\not\mid\mid_R$ *grayscale*.

Suppose the *movie* object m is displayed, whose QoS is ⟨30 [fps], 256 [colors] ⟩. The application can obtain the same QoS by performing *mediascale* and *reduce* in any order. In any case, the application can get a state with 15 fps and 16 colors. *mediascale* $\parallel$ *reduce*. *reduce* $\mid\mid\mid$ *mediascale* since *background* is optional. Suppose that RoS R shows "an application is interested in the color of the car". *reduce* $\mid_R$ *mediascale* but *add* $\not\mid_R$ *grayscale*. □

Suppose an application does not care how colorful movies are. A method *update* changes a colored *movie* to a monochromatic one. The application *displays* a colored *movie* object m. If *update* is performed on the state m of the *movie* object, the monochromatic version of m is seen. Since the application is not interested in the color, both of the versions are considered to satisfy RoS R. Hence, $Q([display(m)]) \cap Q([update \circ display(m)]) \succeq R$ and $Q(display \circ update(m)) = Q(update \circ display(m))$. *display* and *update* are RoS-compatible ($\mid_R$). However, *display* and *update* semantically conflict because $Q([update \circ display(m)]) \neq Q([display(m)])$. □

Let *State*, *Sem*, *RoS*, *QoS*, *Sem-RoS*, and *Sem-QoS* be sets of possible state-, semantically, RoS-, QoS-, semantically RoS-, and semantically QoS-compatible relations on methods. Let c be a family of *State*, *Sem*, *RoS*, *QoS*, *Sem-RoS*, and *Sem-QoS*. Figure 4 shows a Hasse diagram on the sets in c.

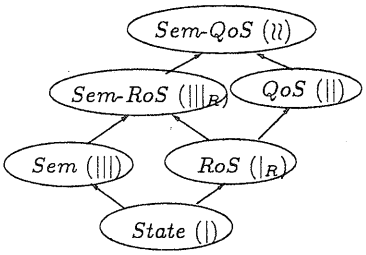

Figure 4: Compatibility relations.

[**Theorem**] The Hasse diagram shown in Figure 4 holds for the α-compatibility relations. □

The Hasse diagram shown in Figure 4 is isomorphic with Figure 3.

4 Synchronization

4.1 Traditional locking protocol

Suppose a pair of transactions T_i and T_j issue methods op_t and op_u to an object o, respectively. Here, T_i *precedes* T_j ($T_i \rightarrow T_j$) iff op_t and op_u conflict and op_t is performed on the object o before op_u. A collection of the transactions $T_1, \ldots, T_m$ are *serializable* iff either $T_i \rightarrow T_j$ or $T_j \rightarrow T_i$ for every pair of transactions T_i and T_j, i.e. the transactions are totally ordered in the precedent relation "$\rightarrow$". In order to do that, an object o is locked before a method op_t is performed on o. If o is already locked for a method op_u conflicting with op_t, op_t blocks until the lock held by op_u is released. On the other hand, every pair of compatible methods can be performed on the object o in any order. An object o is locked in a mode $\mu(op_t)$ before op_t is performed. $\mu(op_t)$ *conflicts* with $\mu(op_u)$ if op_t conflicts with op_u. Otherwise the modes are *compatible*. If o is locked in a mode conflicting with $\mu(op_t)$, op_t blocks. Otherwise, op_t is performed. It is well known that a collection of transactions are *serializable* if every transaction is two-phase locked [1].

Multiple conflicting methods cannot be concurrently performed on an object. It is still question whether or not multiple compatible methods can be concurrently performed. In some systems, some compatible methods like a pair of *display* methods can be concurrently performed on the *movie* object. On the other hand, some compatible methods cannot be concurrently performed, i.e. the methods are to be mutually exclusively performed. For example, *add* and *subtract* are compatible on a *counter* object but cannot be concurrently performed. *Reduce* and *mediascale* are RoS-compatible (*reduce* $|_R$ *mediascale*) but cannot be concurrently performed on the *movie* object as shown in Example 2. It is critical to definitely separate the lock concept to a pair of orthogonal concepts, locks for serialization and for mutual exclusion.

4.2 α-conflicting relations

If a pair of methods op_t and op_u α-conflict in an object o ($op_t \not\diamond_\alpha op_u$), the result obtained by performing op_t and op_u depends on the computation order of op_t and op_u. In the *movie* object m discussed in Example 2, the method *reduce* is *RoS*-compatible with *mediascale* on some RoS R (*reduce* $|_R$ *mediascale*). This means *reduce* and *mediascale* can be performed on the object m in any order as far as a given RoS R is satisfied. However, *reduce* and *mediascale* cannot be concurrently reformed, i.e. mutually exclusive. On the other hand, a pair of *display* methods can be performed in any order since *display* is compatible with itself. In addition, multiple *display* methods can be concurrently performed because multiple transactions can view the *movie* object m at the same time. The traditional concurrency control theories [1] assume every pair of conflicting methods are mutually exclusive while compatible methods can be concurrently performed. However, some pair of compatible methods cannot necessarily be concurrently performed on a multimedia object. For example, unless some system allows multiple transactions to simultaneously view the *movie* object m, at most one *display* can be performed on m at a time. Furthermore, some pair of conflicting methods can be concurrently performed on the multimedia object while it is critical to consider which method is started before the other. For example, let us consider a *blackboard* object b which supports a method *paint*. By using the *paint* method, some area in the *blackboard* is painted. Suppose that an each area in the blackboard is overwritten by the *paint* method. If a pair of *paint* methods pnt_1 and pnt_2 are issued to the *blackboard* object b, it is critical to make clear which method pnt_1 or pnt_2 is performed before the other method if the areas painted intersect on b. After one of pnt_1 and pnt_2 is started on b, the other one can be concurrently performed.

We newly define an *α-mutually exclusive* relation among methods as follows.

[**Definition**] A method op_t is *α-mutually exclusive* with op_u on a class c iff neither $op_t \parallel op_u$ is α-compatible with $op_t \circ op_u$ ($op_t \parallel op_u \diamond_\alpha op_t \circ op_u$) nor $op_t \parallel op_u \diamond_\alpha op_t \circ op_u$. □

If op_t is *α-mutually exclusive* with op_u, op_t and op_u cannot be concurrently performed in order to keep the object of c consistent with respect to $\diamond_\alpha$. Unless op_t is α-mutually exclusive with op_u, a concurrent computation of op_t and op_u results in the same as any serial computation of $op_t \circ op_u$ and $op_u \circ op_t$. The α-mutually exclusive relation is symmetric and transitive.

We define two new types of α-conflicting relations :

[**Definition**] Let op_t and op_u be methods of a class c.

1. op_t *strongly α-conflicts* with op_u on c iff op_t α-conflicts with op_u and op_t is α-mutually exclusive with op_u.
2. op_t *weakly α-conflicts* with op_u on c iff op_t α-conflicts with op_u and op_t is not α-mutually exclusive with op_u. □

4.3 QoS based lock modes

The following orthogonal types of lock modes for a method op_t of an object o are introduced with respect to the α-compatibility and α-mutually ex-

clusive relations :

1. *α-serialization* lock mode $\sigma_\alpha(op_t)$, and
2. *α-mutually exclusive* lock mode $\mu_\alpha(op_t)$.

The serialization locks are used to serialize a collection of conflicting methods issued by different transactions. That is, the lock can be used to decide which method to be started before the other. If an object is locked in an α-serialization mode conflicting with $\sigma(op_t)$, op_t blocks. The mutually exclusive locks are used to make methods mutually exclusively performed on an object.

The conflicting relation among *α-serialization* and *α-mutually exclusive* lock modes is defined as follows:

[Theorem] For every pair of methods op_t and op_u supported by an object o,

1. $\sigma_\alpha(op_t)$ *conflicts with* $\sigma_\alpha(op_u)$ and $\mu_\alpha(op_t)$ *conflicts with* $\sigma_\alpha(op_u)$ if op_t strongly α-conflicts with op_u.
2. $\sigma_\alpha(op_t)$ *conflicts with* $\sigma_\alpha(op_u)$ and $\mu_\alpha(op_t)$ is *compatible with* $\mu_\alpha(op_u)$ if op_t weakly α-conflicts with op_u.
3. $\mu_\alpha(op_t)$ *conflicts with* $\mu_\alpha(op_u)$ if op_t is α-mutually exclusive with op_u. □

Let τ_1 and τ_2 be lock modes. τ_1 is compatible with τ_2 ($\tau_1 \diamond \tau_2$) iff τ_1 does not conflict with τ_2. For example, the mutually exclusive mode $\mu_\alpha(display)$ is compatible with $\mu_\alpha(display)$ while the serialization mode $\sigma_\alpha(display)$ is compatible with $\sigma_\alpha(display)$. $\sigma_R(reduce)$ is compatible with $\sigma_R(mediascale)$ on RoS R since *reduce* $|||_R$ *mediascale*. However, $\mu_R(reduce)$ is not compatible with $\mu_R(mediascale)$ on R since *reduce* and *mediascale* cannot be concurrently performed. Furthermore, $\mu_R(reduce)$ is not compatible with $\mu_R(mediascale)$. Every type of conflicting relation is assumed to be symmetric but not transitive.

Suppose that an object o is locked for a method op_t and another method op_u is issued to the object o. If $\sigma_\alpha(op_u)$ conflicts with $\sigma_\alpha(op_t)$, op_t blocks until op_u terminates. Suppose an object x supports a pair of methods op_1 and op_2 on x and another object y supports op_3 and op_4 on y. A transaction T_1 issues op_1 to x and op_3 to y. Another transaction T_2 issues op_2 to x and op_4 to y. First, suppose op_1 is α-compatible with op_2 ($op_1 \diamond_\alpha op_2$) and op_3 $\diamond_\alpha$ op_4. Here, $\sigma_\alpha(op_1)$ is compatible with $\sigma_\alpha(op_2)$ ($\sigma_\alpha(op_1) \diamond \sigma_\alpha(op_2)$) and $\sigma_\alpha(op_3) \diamond \sigma_\alpha(op_4)$. op_1 and op_2 can be performed on the object x and op_3 and op_4 on y in any order. For example, op_1 is performed after op_2 on x and op_4 is performed after op_3 on y.

Next, suppose $\mu_\alpha(op_1)$ conflicts with $\mu_\alpha(op_2)$ but $\mu_\alpha(op_3)$ is compatible with $\mu_\alpha(op_4)$. op_2 can be started after op_1 completes. op_1 and op_2 cannot be concurrently performed. However, op_3 and op_4 can be concurrently performed on the object y because $\mu_\alpha(op_3)$ is compatible with $\mu_\alpha(op_4)$.

[Theorem] Suppose a method op_t strongly α-conflicts with op_u. A mutually exclusive mode $\mu_\alpha(op_t)$ conflicts with $\mu_\alpha(op_u)$ if a serialization mode $\sigma_\alpha(op_t)$ conflicts with $\sigma_\alpha(op_u)$. □

If a method op_t α-conflicts with another method op_u on an object o, op_t and op_u cannot be concurrently performed. For example, a pair of the methods *add* and *mediascale* *QoS*-conflict and cannot be performed on the *movie* object m at the same time. That is, $\mu_{QoS}(add)$ conflicts with $\mu_{QoS}(mediascale)$.

If a transaction T issues a method op_t in the α-conflicting relation to an object o, o is locked according to the following protocol.

[Locking protocol]

1. The transaction T first issues a serialization lock request $\sigma_\alpha(op_t)$ to the object o.
2. If the object o is not locked in any mode conflicting with $\sigma_\alpha(op_t)$, o is locked in $\sigma_\alpha(op_t)$ and then the lock mode is tried to be converted in a mode $\mu_\alpha(op_t)$.
3. If the lock mode is converted, op_t is ready to be performed on the object o.
4. Otherwise, op_t blocks. □

4.4 Relation among lock modes

Figure 4 shows how compatibility, i.e. conflicting relations of methods of an object o are related. Here, *State*, *QoS*, *RoS*, *Sem*, *Sem-QoS*, and *Sem-RoS* indicate sets of possible *State*-, *QoS*-, *RoS*-, semantically, and semantically *QoS*- and *RoS*-conflicting relations of a class c, respectively. Let C be a family of the sets {*State*, *QoS*, *RoS*, *Sem*, *Sem-QoS*, *Sem-RoS*}. The Hasse diagram as shown in Figure 4 holds for the α-conflicting relations in C. For example, a method op_t *QoS*-conflicts with another method op_u if op_t *Sem-conflicts* with op_u. Hence, *Sem* $\subseteq$ *QoS*.

Let α_1 and α_2 be two types of conflicting relations. α_1 *dominates* α_2 ($\alpha_1 \succ \alpha_2$) iff $\alpha_1 \supseteq \alpha_2$ in Figure 4. For example, *Sem-QoS* $\prec$ *QoS* $\prec$ *State* because *State* $\rightarrow$ *QoS* $\rightarrow$ *Sem-QoS* in Figure 4. α_1 and α_2 are *uncomparable* ($\alpha_1 \parallel \alpha_2$) if neither $\alpha_1 \succ \alpha_2$ nor $\alpha_2 \succ \alpha_1$. In Figure 4, *Sem* $\parallel$ *RoS*. Let $C_\alpha(\tau)$ be a set of lock modes which are compatible with a lock mode τ with respect to an α-compatible relation ($\diamond_\alpha$). The following property holds on the dominant relation "$\prec$" :

[Theorem] For every lock mode τ and every pair of conflicting relations α_1 and α_2, $C_{\alpha_1}(\tau) \supseteq C_{\alpha_2}(\tau)$ if $\alpha_1 \succ \alpha_2$. □

[Definition] A lock mode τ_1 on α_1 is *stronger* than another mode τ_2 on α_2 ($\tau_1 \succ \tau_2$) iff $C_{\alpha_1}(\tau_1) \subseteq C_{\alpha_2}(\tau_2)$. □

"$\tau_1 \succ \tau_2$" means that τ_1 conflicts with more number of lock modes than τ_2. The lock mode τ_1 is more restricted than τ_2. For example, *write* $\succ$ *read*.

Let α_1 and α_2 be types of *RoS*-conflicting relations on RoS R_1 and R_2, respectively. We discuss $\alpha_1 \succ \alpha_2$ or $\alpha_1 \prec \alpha_1$. Here, "$R_1 \succ R_2$" (R_1 dominates R_2) means that R_1 shows a higher level of QoS than R_2 as presented before. It is straightforward for the following theorem to hold from the definitions.

[Theorem] Let τ_1 and τ_2 be lock modes on *RoS*-conflicting relations α_1 and α_2, respectively. Let R_1 and R_2 be RoS for τ_1 and τ_2, respectively. τ_1

$\succ \tau_2$ if $R_1 \succ R_2$. □

Suppose that R_1 and R_2 show monochromatic and colored movies, respectively, in Example 2. Let τ_1 and τ_2 be serialization lock modes of a method *grayscale* on RoS R_1 and R_2, respectively, i.e. $\tau_1 = \sigma_{R_1}(grayscale)$ and $\tau_2 = \sigma_{R_2}(grayscale)$. Here, $R_2 \succ R_1$. *grayscale* R_1-conflicts with *add* but is R_2-compatible with *add*. τ_1 is stronger than τ_2 ($\tau_1 \succ \tau_2$) since $C_{R_1}(\tau_1)$ (= {*mediascale*, *reduce*, *add*, *grayscale*}) $\supset$ $C_{R_2}(\tau_2)$ (= {*mediascale*, *reduce*, *grayscale*}).

4.5 Implementation of lock modes

State-conflicting and *QoS*-conflicting relations among methods are defined in defining each object. In this paper, we assume there is one application manipulating distributed objects in the system. The application is composed of multiple transactions. A *Sem*-conflicting relation among methods is defined for each object, based on the semantics of the application. That is, mandatory and optional component classes are defined for a class. Each transaction has its own *RoS*. It consumes plenty of computation power to compare arbitrary RoS instances. Hence, we assume some limited number of RoS instances are specified when the objects are defined in order to reduce the computation overhead. Each object maintains a table showing the conflicting relations among the lock modes. By using the conflicting table, it is decided if the method issued to the object can be performed on the object.

5 Concluding Remarks

We discussed novel types of relations among methods on the basis of QoS and the state of an object, i.e. state-, semantically, QoS-, RoS-, and semantically QoS- and RoS-conflicting relations of methods in the object-based multimedia system. We presented the locking protocol to realize these new types of conflicting relations, where new lock modes, serialization and mutually exclusive modes are introduced. By using the serialization and mutually exclusive locks, we can increase the performance of the system.

In the locking protocol, the lock mode is converted from serialization ones σ to mutually exclusive ones μ is stronger the σ, transactions may be deadlocked. We need unsympathetic modes discussed in the paper [9].

References

[1] Bernstein, P. A., Hadzilacos, V., and Goodman, N., "Concurrency Control and Recovery in Database Systems," *Addison-Wesley*, 1987.

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

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

[4] Gall, D., "MPEG: A Video Compression Standard for Multimedia Applications," *Comm. ACM*, Vol.34, No.4, 1991, pp.46–58.

[5] Garcia-Molina, H. and Salem, K., "Sagas," *Proc. of ACM SIGMOD*, 1987, pp.249–259.

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

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

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

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

[10] MPEG Requirements Group, "MPEG-4 Requirements," ISO/IEC JTC1/SC29/WG11 N2321, 1998.

[11] Object Management Group Inc., "The Common Object Request Broker: Architecture and Specification, Rev2.0," 1995.

[12] Owen, C. B. and Makedon, F., "Computed Synchronization For Multimedia Applications," *Kluwer Academic*, 1999.

[13] Stroustrup, B., "The C++ Programming Language (2nd ed.)," *Addison-Wesley*, 1991.

[14] Yoshida, T. and Takizawa, M., "Model of Mobile Objects," *Proc. of DEXA'96 (Lecture Notes in Computer Science, Springer-Verlag*, No. 1134), 1996, pp. 623–632.