QoS-Based Method for Compensating Multimedia Objects

Motokazu Yokoyama, Katsuya Tanaka, and Makoto Takizawa
Tokyo Denki University
E-mail {moto, katsu, taki}@takilab.k.dendai.ac.jp

Distributed applications like teleconferences using high-speed networks are manipulating multiple multimedia objects. It is critical to discuss what quality of service (QoS) is supported by the multimedia objects. QoS is manipulated in addition to the state of the object. After objects are manipulated, the objects are sometime required to be rolled back in order to undo the manipulation. In traditional ways, a state saved at a checkpoint is restored. However, it is not easy to save the object because the state is large and complex. In addition, it is sufficient for applications to restore a state which supports enough QoS even if the state is not the same as the previous state. In this paper, we discuss a new way where compensating methods are performed to roll back objects.

QoSに基づいたマルチメディアオブジェクトの補償演算

横山 基一　田中 勝也　滝沢 誠
東京電機大学理工学情報システム工学科
E-mail {moto, katsu, taki}@takilab.k.dendai.ac.jp

本論文では、システム環境の変更において、応用の要求を満足するために柔軟性のある分散型システムを構築する方法について論じる。システム内のマルチメディアオブジェクトは、それぞれのオブジェクトが持つサービスのタイプとともに、要求されるサービス品質 (QoS) を提供せねばならない。マルチメディアオブジェクト上で実行された結果を無効にすることが、障害復旧等で必要となる。本論文では、新たにQoS に基づいた補償方法を用いて、オブジェクトを復旧する方法について論じる。ここでは、オブジェクトは以前の状態と同じではないかもしれないが、要求される QoS をサポートできる状態に復旧される。

1 Introduction

Distributed applications are composed of multiple multimedia objects. In traditional systems, checkpoints [2,5,12] and replications [10] are used to make the systems fault-tolerant. Larger and more complex multimedia objects are manipulated and transmitted than the traditional systems. Hence, it takes a longer time to save a state of the object in the log and a larger volume of log storage is required to take a checkpoint. It is also not easy to manipulate multiple replicas of multimedia objects since larger volume of storage and computation overhead are required to store the replicas and to perform the requests on the replicas than traditional data.

It is significant for multimedia objects to support applications with enough quality of service (QoS) [9] like frame rate and number of colors required by the applications. The objects are manipulated only through methods supported by the objects. Not only states but also QoS of objects are changed by methods. A state obtained by reducing QoS of the multimedia object can be taken at the checkpoint if the state satisfies QoS required by the application. By this method, we can reduce a volume of the log and time to take the checkpoint. If an object is faulty, the object can be rolled back to a state which might be different from the previous one but supports the application with sufficient QoS. The state of the multimedia object is large. Hence, if methods performed on the object are logged in stead of storing the state of the object, the size of the log is reduced. The object is rolled back by performing *compensating* methods [7] of the methods performed on the object which are stored in the log. By this method, we can reduce a time to roll back the objects. We newly discuss QoS-based how compensating methods are related in this paper.

In this paper, we first present a system model and QoS in section 2. In section 3, we discuss kinds of relations among methods in multimedia objects. In section 4, we discuss compensating methods.

2 Object-Oriented Model

2.1 Objects

A system is composed of objects which are distributed on computers interconnected by networks. 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. In most object-oriented systems like Java [3] and C++ [11], a term *object* is used to show an instance. From here, let *objects* show *instances* in this paper. A collection $\langle v_1, \ldots, v_m \rangle$ of values is a *state* of the object o where each v_i is a value taken by the attribute A_i $(i = 1, \ldots, m)$. An object is assumed to have exactly one state at a time. A state of a class means a state of an object.

A class c can be composed of other classes $c_1, \ldots, c_n$. Here, each c_i is referred to as a *component* class of the class c. This relation between the class c and c_i is a *part-of* relation. Let $c_i(s)$ denote a projection of a state s of the class c to a subclass c_i.

On receipt of a request of a method op, op is performed on an object o. Then, the response is sent back. Let $op(s)$ and $[op(s)]$ denote a state and response obtained by performing a method op on a state s of an object o, respectively. For example, an *image* object I supports a method *display*. $display(s) = s$ since *display* does not change the state and $[display(s)]$ shows *image* of the state s of the object I displayed on the monitor. Let op_1 and op_2 be methods supported by an object o. $op_1 \circ op_2$ shows that op_2 is performed after op_1 competes, i.e. op_1 and op_2 are serially performed on the object o.

2.2 QoS of object

Applications obtain service of an object o through methods. Each service is characterized by parameters like level of resolution and number of colors. These parameters are referred to as *quality of service* (*QoS*).

A QoS *value* is given in a tuple of values $\langle v_1, \ldots, v_m \rangle$ where each v_i is a value of parameter like frame rate. A QoS *value* v_1 *precedes* another v_2 ($v_1 \succeq v_2$) if v_1 shows better QoS than v_2. For example, $120 \times 100 \preceq 160 \times 120$ [pixels] for the attribute *resolution*. A QoS *value* q_1 *dominates* q_2 ($q_1 \succeq q_2$) iff q_1 shows a better level of QoS than q_2. For example, $\langle 160 \times 120\text{[pixels]}, 1024\text{[colors]}, 15\text{[fps]} \rangle \succeq \langle 120 \times 100, 512, 15 \rangle$. Let S be a set of QoS *values*. A QoS value q_1 is *minimal* in the set S iff there is no QoS value q_2 in S such that $q_2 \preceq q_1$. q_1 is *minimum* iff $q_1 \preceq q_2$ for every q_2 in S. q_1 is *maximal* iff there is no q_2 in S such that $q_1 \preceq q_2$. q_1 is *maximum* iff $q_2 \preceq q_1$ for every q_2 in S. A *least upper bound (lub)* $q_1 \cup q_2$ is some QoS value q_3 in S such that 1) $q_1 \preceq q_3$ and $q_2 \preceq q_3$, and 2) there is no value q_4 in S where $q_1 \preceq q_4 \preceq q_3$ and $q_2 \preceq q_4 \preceq q_3$.

Each state s of an object o supports a QoS value denoted by $Q(s)$. An application requires an object o to support some QoS, named *requirement* QoS (*RoS*). Let r be RoS.

3 Compatible Methods

Suppose a class c is composed of component classes $c_1, \ldots, c_m$ ($m \geq 0$). An application specifies whether each c_i is *mandatory* or *optional* for the class c. Every object o of the class c is required to include an object o_i of a mandatory class c_i. If c_i is optional, the object o may not include any object of c_i. There are the following equivalent relations among 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, i.e. $Q(s_t) \cup Q(s_u) \preceq Q(s)$.
- 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.

Let $\Box_\alpha$ show an α-equivalent relation and α shows some equivalent relation. For example, $\Box_{QoS}$ shows "$\approx$" and $\Box_{Sem}$ indicates "$\equiv$". Figure 1 indicates a Hasse diagram showing the properties of the equivalent relations. Here, *State, Sem, RoS, QoS, RoS, Sem-QoS, Sem-RoS* stand for sets of possible *state-*, *semantically, QoS-, RoS-*, *semantically QoS-*, and *semantically RoS-*equivalent relations, respectively. Here, $\alpha \rightarrow \beta$ shows that β is a subset of α, i.e. $\alpha \subseteq \beta$. That is, $s_t \Box_\beta s_u$ if $s_t \Box_\alpha s_u$. For example, $s_t \equiv s_u$ if $s_t - s_u$.

For a pair of methods op_t and op_u of a class c, "$op_t \Box_\alpha op_u$" shows that $op_t(s) \Box_\alpha op_u(s)$ for every state s of c. For example, $op_t \equiv op_u$ (op_t is *semantically equivalent* with op_u) if $op_t(s) \equiv op_u(s)$ for every state s of c. $op_t \cong op_u$ (op_t is *semantically QoS-equivalent* with op_u) if $op_t \equiv op_u$ or $op_t \approx op_u$. These equivalent relations hold for a pair of sequences of methods, i.e. $S_1 \Box_\alpha S_2$. For example, let S_1 be $op_1 \circ op_2$ and S_2 be op_3. $S_1 \equiv S_2$, i.e. $op_1 \circ op_2 \equiv op_3$ iff $op_1 \circ op_2(s) \equiv op_3(s)$ for every state s of a class c. In addition, ϕ shows an empty sequence of methods. $op \Box_\alpha \phi$ iff $op(s) \Box_\alpha s$ for every state s of the class c.

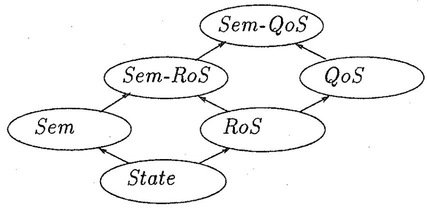


Figure 1: Hasse diagram.

In the traditional theories [1,7], a method op_t is *compatible* with another method op_u on a class c iff the result obtained by performing op_t and op_u is independent of the computation order. Otherwise, op_t *conflicts* with op_u. There are the following compatible relations among 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 ($op_t \wr\wr op_u$) 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 $op_t \circ op_u \equiv_R op_u \circ op_t$.

Here, let "*α-compatible* relation" $\Diamond_\alpha$ show some type of the compatible relations defined here. $\alpha \in \{$*State*, *QoS*, *Semantically*, *RoS*, *Sem-QoS*, *Sem-RoS*}. For example, $op_t \diamond_{Sem} op_u$ stands for "$op_t \equiv op_u$". op_t *α-conflicts* with op_u unless op_t is α-compatible with op_u. For example, op_t *QoS-conflicts* with op_u unless op_t is *QoS-compatible* with op_u. Let *State, Sem, QoS, RoS, Sem-QoS,* and *Sem-RoS* be sets of possible *state, semantically-, QoS-, RoS-, semantically QoS-,* and *semantically RoS-compatible* relations, respectively. The same relations among the sets as shown in Figure 1 hold. op_t *α-conflicts* with op_u unless $op_t \diamond_\alpha op_u$.

4 Compensation

4.1 Compensating methods

In traditional systems like database systems [1], a state of a system is saved into a log at a checkpoint. If the system is faulty, the state stored in the log is restored in the system and then the system is restarted. In multimedia systems, objects are larger and more complex than simple objects like files and tables.

In another way, methods performed on each object are stored in the log instead of storing the state of the object. A method which removes the effect done by the method performed is a *compensating* method [7]. For example, suppose an *increment* method is performed on a *counter* object. If a *decrement* method is performed, the *counter* object can be restored. *decrement* is a compensating method of *increment*. *increment* is also referred to as *compensated* by *decrement*. Thus, the object is restored by compensating the methods stored in the log.

A method op_u is a *compensating* method of another method op_t on a class c if $op_t \circ op_u(s) = s$ for every state s of the class c [7]. Let s_1 be a state obtained by performing op_t on a state s of an object o of the class c, i.e. $s_1 = op_t(s)$. Here, the object o can be rolled back to the initial state s from the state s_1 if a compensating method of op is performed on s_1.

There are the following relations among a pair of methods op_t and op_u of a class c according to the α-equivalent relation $\Box_\alpha$:

- op_u *state-compensates* op_t iff $op_t \circ op_u = \phi$.
- op_u *QoS-compensates* op_t iff $op_t \circ op_u \approx \phi$.
- op_u *RoS-compensates* op_t on R iff $op_t \circ op_u =_R \phi$.
- op_u *semantically compensates* op_t iff $op_t \circ op_u \equiv \phi$ [Figure2(1)].
- op_u *semantically QoS-compensates* op_t iff $op_t \circ op_u \cong \phi$.
- op_u *semantically RoS-compensates* op_t on R iff $op_t \circ op_u \equiv_R \phi$ [Figure2(2)].

Here, let an *"α-compensation"* method show a type of the compensating methods presented here, where $\alpha \in$ {*State, Sem, QoS, RoS, Sem-QoS, Sem-RoS*}.

[Theorem] The Hasse diagram shown in Figure 1 holds for the α-compensating relations. □

[Example] Suppose a *movie* object C supports a method *divide2* by which C is divided into three

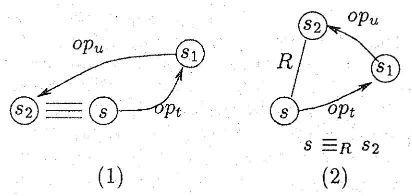


Figure 2: Compensating methods.

subobjects A'', B'', and AB in addition to the methods *merge* and *delete* [Figure 3]. A state s_1 of the object C is composed of two component objects A and B. A'' and B'' show the *content* parts of A and B, respectively, which are monochromatic in a s_3. AB includes the *advertisement* objects of A and B. Let s_3 denote a state where the objects A'', B'', and AB are obtained from A and B existing in a state s_1. $s_1 \neq s_3$. Furthermore, $Q(s_2) \succeq Q(s_1)$. If a RoS R indicates the monochromatic quality, $Q(s_3) \succeq R$. Hence, *divide2* is a *semantically RoS-compensating* method of the method *merge* on R. By performing *divide2* after *merge* on s_1, s_3 is obtained where $s_1 \equiv_R s_3$. □

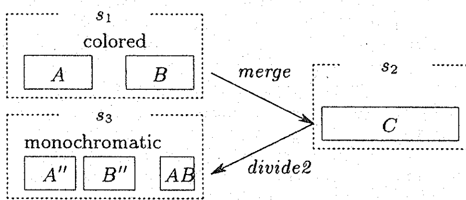


Figure 3: Semantically RoS-compensating method.

Let ($\sim_\alpha op$) denote an *α-compensating* method of a method op with respect to the α-compensating relation. For example, ($\sim_{Sem} op$) shows a *Sem-compensating* method of the method op. From the theorem, ($\sim_\alpha op$) is ($\sim_\beta op$) if $\alpha \rightarrow \beta$. For example, ($\sim_{State} op$) is ($\sim_{Sem} op$). $op \circ (\sim_{State} op)(s) = s$, $op \circ (\sim_{Sem} op)(s) = s'$, and $s \equiv s'$ [Figure 4]. Hence, $(\sim_{State} op) \equiv (\sim_{Sem} op)$. That is, if op' is a *State-compensating* method of op, op is also a *Sem-compensating* method of op. More precisely, the following theorem holds for α- and β- compensating methods.

[Theorem] $(\sim_\alpha op) \Box_\beta (\sim_\beta op)$ iff $\alpha \rightarrow \beta$. □

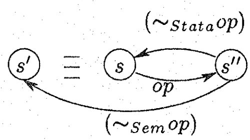


Figure 4: Compensating methods.

4.2 Compensating sequence of methods

Suppose that a sequence of methods op_1, ..., op_n are performed on a state of an object o, i.e. $op_1 \circ \ldots \circ op_n$. First, let us consider the *State*-equivalent compensation. Suppose op_2 is performed after op_1 on an object o, i.e. $op_1 \circ op_2$. Here, a compensating method $(\sim_{State} op_2)$ is performed, i.e. $op_1 \circ op_2 \circ (\sim_{State} op_2) - op_1$. Then, $(\sim_{State} op_1)$ is performed, i.e. $op_1 \circ op_2 \circ (\sim_{State} op_2) \circ (\sim_{State} op_1) - op_1 \circ (\sim_{State} op_1) - \phi$. For example, *delete* is $(\sim_{State} append)$ and *replace* is $(\sim_{State} replace)$. $\sim_{State}(append \circ replace)$ is *state-equivalent* with $(-)$ $(\sim_{State} replace) \circ (\sim_{State} append) = replace \circ delete$. Thus, the effect on the object o can be removed by performing the compensating methods of op_1 and op_2, i.e. $(\sim_{State} op_2) \circ (\sim_{State} op_1)$. That is, $\sim_{State}(op_1 \circ op_2)$ is *state*-equivalent with $(\sim_{State} op_2) \circ (\sim_{State} op_1)$, i.e. $\sim_{State}(op_1 \circ op_2) - (\sim_{State} op_2) \circ (\sim_{State} op_1)$. Thus, $\sim_{State}(op_1 \circ \ldots \circ op_n) - (\sim_{State} op_n) \circ \ldots \circ (\sim_{State} op_1)$.

We discuss how an *α-compensation* $\sim_\alpha(op_1 \circ \ldots \circ op_n)$ of a sequence $op_1 \circ \ldots \circ op_n$ is equivalent with a sequence of compensating methods $(\sim_{\alpha_n} op_n) \circ \ldots \circ (\sim_{\alpha_1} op_1)$, i.e. $\sim_\alpha(op_1 \circ \ldots \circ op_n) \Box_{\alpha_0} (\sim_{\alpha_n} op_n) \circ \ldots \circ (\sim_{\alpha_1} op_1)$ where α_0, α_1, ..., α_n are some types of the compensating relations. In this paper, we consider a case $\alpha_0 = \alpha$ for simplicity.

Before discussing how to compensate a sequence of methods, we discuss types of methods with respect to what the methods change. There are two types of methods, one to change the state of the object and the other to change QoS of the object. For example, *grayscale* is a method which changes only QoS of the *movie* object. On the other hand, *merge* and *delete* are methods to change the state, but does not change QoS of the *movie* object. Next point is concerned with what component of the object each method changes. There are two types of component classes, mandatory and optional ones as discussed before. Hence, there are two types of methods, one to change the mandatory component object and the other not to change the mandatory object. The former one is named *semantical* method. The other one is *formal* one. According to the properties of the methods, the methods are classified into types shown in table 1. Here, let $\mu(op)$ show a type of a method op, i.e. $\mu(op) \in \{S, SM, SO, Q, QM, QO, R, RM, RO\}$.

Let α_1 and α_2 be a pair of compensating relations. Suppose that a pair of methods op_1 and op_2 are performed on an object o, i.e. $op_1 \circ op_2$. We discuss how to compensate $op_1 \circ op_2$. We discuss how $\sim_\alpha(op_1 \circ op_2)$ and $(\sim_{\alpha_2} op_2) \circ (\sim_{\alpha_1} op_1)$ are related, i.e. $\sim_\alpha(op_1 \circ op_2) \Box_\alpha (\sim_{\alpha_2} op_2) \circ (\sim_{\alpha_1} op_1)$ on the basis of method types $\mu(op_1)$ and $\mu(op_2)$. $\alpha, \alpha_1, \alpha_2 \in \{State, Sem, QoS, RoS, Sem\text{-}QoS, Sem\text{-}RoS\}$. The following matrixes show how α_1- and α_2-compensating relations are related with α-equivalent relation. Here, each entry $M_i(\mu_1, \mu_2)$ shows a condition which $\sim_\alpha(op_1 \circ op_2) \Box_\alpha (\sim_{\alpha_2} op_2) \circ (\sim_{\alpha_1} op_1)$ holds for a type μ_1 of op_1 and μ_2 of op_2. For example, let us consider $M_1(SM, S)$. Here, $\sim_{Sem}(op_1 \circ op_2) \equiv (\sim_{State} op_2) \circ (\sim_{Sem} op_1)$ holds if $\mu(op_1) = SM$ and $\mu(op_2) = S$. In the matrixes, $\alpha_j = \phi$ shows "$(\sim_{\alpha_j} op_j)$ is not performed". For example, if op_1 is an SO type and op_2 is an S type, $M_1(SO, S) =$ B, i.e.

Table 1: Types of methods.

type	what to be changed in an object
S	state.
SM	state of the mandatory objects.
SO	state of the optional objects.
Q	QoS.
QM	QoS of the mandatory object.
QO	QoS of the optional object.
R	Q method such that $op_t(s) \succeq R$ for every state s of c and some RoS R.
RM	QM method where $c_i(op_t(s)) \succeq R$ for every state s of mandatory component class c_i of c and some RoS R.
RO	QO method such that $c_i(op_t(s)) \succeq R$ for every state s of optional component class c_i of c.

$\sim_{Sem}(op_1 \circ op_2) \equiv (\sim_{State} op_2)$. Since op_1 updates only optional component object, $op_1(s) \equiv s$ for every state s, i.e. $op_1 \equiv \phi$ [Figure 5]. Hence, $(\sim_\alpha op_1)$ is not required to be performed.

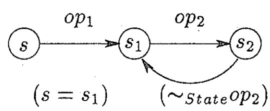
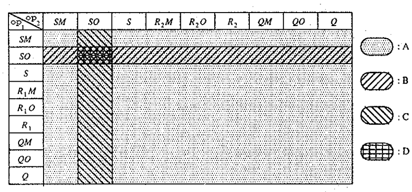


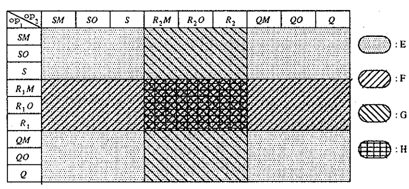
Figure 5: Compensation.

M_1: $\alpha =$ "$\equiv$".



A: $\alpha_1, \alpha_2 \in \{State, Sem\}$.
B: $\alpha_1 = \phi \ \wedge \ \alpha_2 \in \{State, Sem\}$.
C: $\alpha_1 \in \{State, Sem\} \ \wedge \ \alpha_2 = \phi$.
D: $\alpha_1 = \alpha_2 = \phi$.

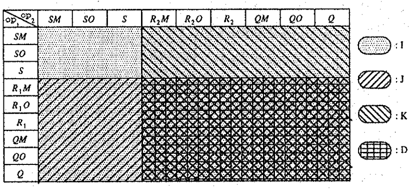
M_2: $\alpha =$ "$-_R$".



E: $\alpha_1, \alpha_2 \in \{State, RoS(\succeq R)\}$.
F: $\alpha_1 = \phi \ \wedge \ \alpha_2 \in \{RoS(\succeq R), State\} \ \wedge \ R_2 \cap Q(op_1(s)) \succeq R$.

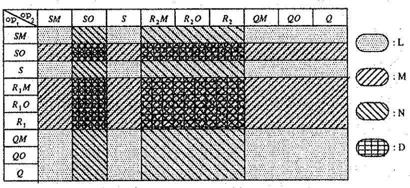
G: $\alpha_1 \in \{State, RoS(\succeq R)\} \wedge \alpha_2 = \phi$
$\wedge\ R_1 \cap Q(op_2(s)) \succeq R$.
H: $\alpha_1 = \alpha_2 = \phi \wedge R_1 \cap R_2 \succeq R$.

M_3: $\alpha =$ "$\approx$".



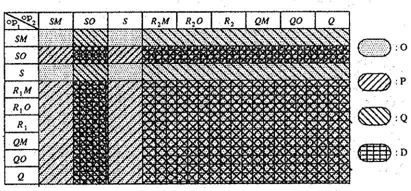
I: $\alpha_1 = \alpha_2 \in \{QoS, RoS, State\}$.
J: $\alpha_1 = \phi \wedge \alpha_2 \in \{QoS, RoS, State\}$.
K: $\alpha_1 \in \{State, QoS, RoS\} \wedge \alpha_2 = \phi$.

M_4: $\alpha =$ "$\equiv_R$".



L: $\alpha_1, \alpha_2 \in \{State, Sem, RoS, Sem\text{-}RoS\}$
$\wedge\ R_1 \cap R_2 \succeq R$.
M: $\alpha_2 \in \{State, Sem, RoS, Sem\text{-}RoS\}$
$\wedge\ \alpha_1 = \phi \wedge R_2 \cap Q(op_1(s)) \succeq R$.
N: $\alpha_1 \in \{State, Sem, RoS, Sem\text{-}RoS\}$
$\wedge\ \alpha_2 = \phi \wedge R_1 \cap Q(op_2(s)) \succeq R$.

M_5: $\alpha =$ "$\cong$".



O: $\alpha_1, \alpha_2 \in \{State, Sem, QoS, RoS, Sem\text{-}QoS, Sem\text{-}RoS\}$.
P: $\alpha_2 \in \{State, Sem, QoS, RoS, Sem\text{-}QoS, Sem\text{-}RoS\} \wedge \alpha_1 = \phi$.
Q: $\alpha_1 \in \{State, Sem, QoS, RoS, Sem\text{-}QoS, Sem\text{-}RoS\} \wedge \alpha_2 = \phi$.

It is straightforward for the following theorem to hold from the discussions held here.
[Theorem] $\sim_\alpha(op_1 \circ op_2) \Box_\alpha (\sim_{\alpha_2} op_2) \circ (\sim_{\alpha_1} op_1)$ holds iff one of the relations shown in Table 2 holds where "-" of α means any one of $\{State, Sem, QoS, RoS, Sem\text{-}QoS, Sem\text{-}RoS\}$ and "α" of α_i means "$\alpha_i = \alpha$". □

Table 2: Compensation.

α_1	α_2	α
α	α	-
State	*State*	-
State	α	-
α	*State*	-
$Sem \wedge (op_1 \equiv \phi)$	α	-
α	$Sem \wedge (op_2 \equiv \phi)$	-
$RoS \wedge (op_1 - \phi)$	α	-
α	$RoS \wedge (op_2 - \phi)$	-
State	*Sem-RoS*	*Sem-RoS*
Sem-RoS	*State*	*Sem-RoS*
RoS	*Sem*	*Sem-RoS*
Sem	*RoS*	*Sem-RoS*

4.3 Reduced compensating sequence

Let us consider a sequence of two methods, $op_1 \circ op_2$. Here, suppose op_1 is *state-compatible* with op_2, i.e. $op_1 \mid op_2$. Here $(op_1 \circ op_2) - (op_2 \circ op_1)$. Hence, $op_1 \circ op_2$ can be also compensated by a sequence $(\sim_{State} op_1) \circ (\sim_{State} op_2)$ while compensated by $(\sim_{State} op_2) \circ (\sim_{State} op_1)$. This means $(\sim_{State} op_1) \circ (\sim_{State} op_2) - (\sim_{State} op_2) \circ (\sim_{State} op_1)$. Thus, the following theorem holds:

[Theorem] For a pair of methods op_1 and op_2, $op_1 \Diamond_\alpha op_2$ iff $(\sim_\alpha op_1) \Diamond_\alpha (\sim_\alpha op_2)$. □

That is, for a pair of α-*compatible* methods op_1 and op_2, the α-*compensating* methods of op_1 and op_2 are also α-*compatible*.

By using this α-compatibility relation of the methods, we can exchanging the computation order of the methods. Let S be a sequence $S_1 \circ op_1 \circ S_2 \circ op_2 \circ S_3$ of methods where S_1, S_2, and S_3 are subsequences of methods and op_1 and op_2 are methods. Let S' be a sequence $S_1 \circ op_2 \circ S_2 \circ op_1 \circ S_3$. Here, if op_1 is α-*compatible* with op_2 and every method op in S_2 is α-*compatible* with op_1 and op_2, i.e. $op_1 \Diamond_\alpha op_2$, $op \Diamond_\alpha op_1$, and $op \Diamond_\alpha op_2$ for every method op in S_2, $S \Box_\alpha S'$ (S is α-*equivalent* with S'). Here, it is straightforward for the following theorem to hold:

[Theorem] $\sim_\alpha(S_1 \circ op_1 \circ S_2 \circ op_2 \circ S_3) \Box_\alpha (\sim_\alpha S_3) \circ (\sim_\alpha op_1) \circ (\sim_\alpha S_2) \circ (\sim_\alpha op_2) \circ (\sim_\alpha S_1)$. □

The methods *add* and *grayscale* are *RoS-compatible*, i.e. *add* $|_R$ *grayscale*. Suppose *add* is performed before *grayscale*, i.e. *add* $\circ$ *grayscale*. This sequence is *RoS-compensated* by $(\sim_{RoS} grayscale) \circ (\sim_{RoS} add)$. However, it takes a shorter time to perform the compensating method $(\sim_{RoS} grayscale)$ after removing a car added by *add*, i.e. $(\sim_{RoS} add)$. Hence, *add* $\circ$ *grayscale* can be more efficiently compensated by $(\sim_{RoS} add) \circ (\sim_{RoS} grayscale)$, i.e. $add \circ grayscale \circ (\sim_{RoS} add) \circ (\sim_{RoS} grayscale) -_R \phi$.

Next, let us consider how to reduce compensating methods to be performed to compensate a sequence of methods. Suppose that a *car* object c is deleted after added, i.e. *add* $\circ$ *delete* is performed. Since *add* $\circ$ *delete* $- \phi$ holds,

$(\sim_{State} delete) \circ (\sim_{State} add)$ is not required to be performed. Next, suppose a *paint* method $paint_1$ which paints *red* is performed after painting *yellow* by $paint_2$. Since the colors are over painted, $paint_2 \circ paint_1$ brings the same result obtained by performing only $paint_1$. That is, $paint_2 \circ paint_1 - paint_1$.

There are following relations:

1. A method op_t is α-*identical* iff $op_t \sqsupset_\alpha \phi$.
2. A sequence S is α-*identical* iff $S \sqsupset_\alpha \phi$.
3. A method op_t α-*absorbs* another method op_u iff $op_u \circ op_t \sqsupset_\alpha op_t$.
4. A sequence S_1 α-*absorbs* another sequence S_2 iff $S_2 \circ S_1 \sqsupset_\alpha S_1$.

A sequence $add \circ delete$ is *State-identical*. A method *paint State-absorbs* another *paint*.

[Example]

- A method add_2 add a mandatory object *car* and an optional object *background*. A method $delete_2$ deletes *background*. Here, a sequence $add_2 \circ delete_2$ is *Sem-identical*.
- Suppose a video object which supports QoS 20[fps]. A method *increase* is a method which changes the frame rate to 30[fps]. A method *decrease* is a method which changes frame rate to 15[fps]. Here, a sequence $increase \circ decrease$ is *QoS-identical*.
- If there is RoS = 15[fps], a sequence $increase \circ decrease$ is *RoS-identical*.
- A method add_3 adds an optional object *background*. Here, a sequence $add_3 \circ decrease$ is *Sem-QoS-identical*.
- If there is RoS = 15[fps], a sequence $add_3 \circ decrease$ is *Sem-RoS-identical*.
- A method $paint_3$ paints a *car* and a *background blue*. A method $paint_4$ paints *background green*. Here, $paint_3$ *Sem-absorbs* $paint_4$.
- In a colored video object, a method *color* is a method which colors a *car* object. A method *grayscale* is a method which changes all the object black and white. Here, *color QoS-absorbs grayscale*.
- If there is RoS = colored car, *color RoS-absorbs grayscale*.
- $paint_4$ *Sem-QoSabsorbs grayscale*.
- If there is RoS = colored car, $paint_4$ *Sem-QoS-absorbs grayscale*.

Let S be a sequence $S_1 \circ S_2 \circ S_3$ of methods.

- If s_2 is α-*identical*, $(\sim_\alpha S) \sqsupset_\alpha \sim_\alpha (S_1 \circ S_3)$.
- If s_2 is α-*absorbs* s_1, $(\sim_\alpha S) \sqsupset_\alpha (\sim_\alpha S_3)$.

5 Concluding Remarks

In the multimedia systems, QoS of an object is manipulated in addition to the state of the object. In this paper, we discussed how the methods manipulate QoS of the object. We defined semantically, QoS, RoS, semantically QoS, and semantically RoS equivalent and compatible relations among methods of multimedia objects. By using the relations, we defined compensating methods to be used to undo the works done by the methods. We also made clear how types of compensating methods are related from the QoS point of view.

References

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

[2] Chandy, K. M. and Lamport, L., "Distributed Snapshots: Determining Global States of Distributed Systems," *Comm. ACM*, Vol.3, No.1, 1985, pp.63–75.

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

[4] Kanezuka, T., Higaki, H., Takizawa, M., and Katsumoto, M., "QoS Oriented Flexible Distributed Systems for Multimedia Applications," *Proc. of the 13th Int'l Conf on Information Networking* (*ICOIN-13*), 1999, 7C-4.

[5] Koo, R. and Toueg, S.,"Checkpointing and Rollback-Recovery for Distributed Systems," *IEEE Trans. on Software Engineering*, Vol.SE-13, No.1, 1987, pp.23-31.

[6] Yokoyama, M., Tanaka, K., and Takizawa, M.,"QoS-Based Recovery of Multimedia Objects," *Proc. of IEEE Int'l Conf. on Parallel and Distributed Systems (ICPADS-00) Workshops*, pp.43–48, 2000.

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

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

[9] Sabata, B., Chatterjee, S., Davis, M., and Syidir, J. J., "Taxonomy for QoS Specifications," *Proc. of IEEE 3rd Int'l Workshop on Object-Oriented Real-time Dependable Systems* (*WORDS'97*), 1997, pp.100-107.

[10] Schneider, B. F., "Replication Management using the State-Machine Approach," *Distributed Computing Systems*, *ACM Press*, 1993, pp.169-197.

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

[12] Tanaka, K., Higaki, H., and Takizawa, M.,"Object-based Checkpoints in Distributed Systems," *Journal of Computer Systems Science and Engineering*, Vol.13, No.3, 1998, pp.125-133.