Regular Paper

QoS-based Compensation of Multimedia Objects

Motokazu Yokoyama,† Katsuya Tanaka†
and Makoto Takizawa†

In distributed applications, not only the state but also quality of service (QoS) of a multimedia object is manipulated. Like the state, the QoS has to be changed by performing a method. In this paper, we discuss how methods can be undone by performing compensating methods. Novel types of compensating methods are defined to obtain a state and QoS of the objects that satisfy the requirements.

1. Introduction

In distributed multimedia applications, not only the state but also the quality of service (QoS) of a multimedia object is manipulated. In manipulating the object, an application might want to undo a previous manipulation, such as one for interactively designing and implementing action of an application. To take another example, the object may be rolled back because of some fault in it. Suppose that an application changes a colored *movie* object to a monochrome one after adding a *red* car. Here, the *movie* object is monochrome. Next, suppose the application wants to undo the work. According to the traditional ways, the *movie* object is rolled back to the previous one, that is, the colored object without the *car* object. If the application is not interested in how colorful the *movie* object is, only the *car* object can be removed while changing the color. We therefore discuss a novel way to compensate for methods performed on a multimedia object where the QoS and the state of the object are changed so as to satisfy the user's requirements.

In Section 2, we discuss relations among methods. In Section 3, we discuss compensation methods. In Section 4, we classify these methods. In Section 5, we discuss how to compensate for a sequence of methods.

2. QoS-Based Relations of Methods

An object-based system is composed of *classes* and *objects*[4]. A class c is composed of *attributes* $A_1, \ldots, A_m$ ($m \geq 0$) and *methods*. An object o is created from the class c by giving values to attributes. 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 A_i ($i = 1, \ldots, m$). A class c can be composed of *component* classes $c_1, \ldots, c_n$ in a *part-of* relation. Let $c_i(s)$ denote a projection of a state s of the class c to a component class c_i. A state of an object is changed by performing a method *op*. Let $op(s)$ and $[op(s)]$ respectively denote a state and response obtained by performing a method *op* on a state s of an object o. "$op_1 \circ op_2$" shows a serial computation of op_1 and op_2.

Applications obtain service for an object o through methods. Each service is characterized by a *quality of service* (*QoS*). A QoS *value* is a tuple of values $\langle v_1, \ldots, v_m \rangle$ where each v_i is a value of a parameter such as frame rate. 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$160 × 120 [pixels], 1,024 [colors], 15 [fps]$\rangle \succeq \langle 120 \times 100, 512, 15 \rangle$. $q_1 \cup q_2$ shows a least upper bound of q_1 and q_2 on $\succeq$. Let $Q(s)$ be a QoS value of a state s of an object o. $Q(op(s))$ is the QoS obtained through a method *op*. An application requires an object o to support some QoS, named *requirement* QoS (*RoS*).

Suppose a class c is composed of component classes $c_1, \ldots, c_m$ ($m \geq 0$). An application specifies whether each component class c_i is *mandatory* or *optional*. The following relations exist among a pair of states s_t and s_u of a class c[5),6)]:

- s_t is *state-equivalent* to s_u ($s_t - s_u$) iff $s_t = s_u$.
- s_t is *semantically equivalent* to 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* to 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., $Q(s_t) \cup Q(s_u) \preceq Q(s)$.

† Department of Computers and Systems Engineering, Tokyo Denki University

- s_t is *semantically QoS-equivalent* to s_u ($s_t \simeq s_u$) iff $s_t \approx s_u$ or $c_i(s_t) \simeq c_i(s_u)$ for every mandatory component class c_i of c.
- s_t is *r-equivalent* to s_u on RoS r ($s_t \approx_r s_u$) iff $s_t \approx s_u$ and $Q(s_t) \cap Q(s_u) \succeq r$.
- s_t is *semantically r-equivalent* to s_u on RoS r ($s_t \equiv_r s_u$) iff $s_t \approx_r s_u$ or $c_i(s_t) \equiv_r c_i(s_u)$ for every mandatory class c_i of c.

For example, a *movie* class is composed of mandatory classes *car* and *tree* and an optional class *background*. Each state s_i of the *movie* object is composed of component classes *car* c_i, *tree* t_i, and *background* b_i ($i = 1, 2$). $s_1 \simeq s_2$ if c_1 and c_2 show the same car with different QoS and t_1 and t_2 indicate the same tree with different QoS.

Let $\Box_\alpha$ denote an α-equivalent relation where α denotes some equivalent relation. For example, $\Box_{QoS}$ (or $\Box_\approx$) denotes "$\approx$". *State, Sem, QoS, R, Sem-QoS,* and *Sem-R* stand for sets of possible *state, semantically, QoS, R, semantically QoS*, and *semantically R* equivalent relations on states of a class c, respectively. Here, R is $\{\Box_r \mid r$ is a possible QoS$\}$, and *Sem-R* is $\{\Box_{\equiv_r} \mid r$ is a possible QoS$\}$. The relation "$a \to b$" denotes that b is a subset of a. That is, $s_t \Box_b s_u$ if $s_t \Box_a s_u$ for every pair of states s_t and s_u. *State* $\to$ *Sem*, *State* $\to$ *R*, *R* $\to$ *Sem-RR* $\to$ *QoS*, *QoS* $\to$ *Sem-QoS*, *Sem-R* $\to$ *Sem-QoS*.

Let op_t and op_u be a pair of methods of a class c. "$op_t \Box_\alpha op_u$" denotes that $op_t(s) \Box_\alpha op_u(s)$ for every state s of c. ϕ shows an empty sequence of methods. $op \Box_\alpha \phi$ iff $op(s) \Box_\alpha s$ for every state s of c. For example, *display* $- \phi$. Let r_1 and r_2 be a pair of QoS values where $r_1 \succeq r_2$. Here, $\Box_{r_1} \to \Box_{r_2}$ if $r_1 \succeq r_2$. For example, $s_t \approx_{r_1} s_u$ if $s_t \approx_{r_2} s_u$.

In the traditional theories [1),2)], 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.

[**Definition**] For every pair of methods op_t and op_u of a class c, op_t is *α-compatible* with op_u ($op_t \diamond_\alpha op_u$) iff $(op_t \circ op_u) \Box_\alpha (op_u \circ op_t)$ where $\alpha \in \{$*State, QoS, Sem, R, Sem-QoS, Sem-R*$\}$. □

For example, op_t is *semantically compatible* with op_u ($op_t \diamond_\equiv op_u$) iff $(op_t \circ op_u) \equiv (op_u \circ op_t)$. The "*R-compatible* relation" $\diamond_R$ denotes a set $\{\diamond_r \mid r \in R\}$ where R is a set of possible QoS values. op_t *α-conflicts* with op_u ($op_t \not\diamond_\alpha op_u$) unless $op_t \diamond_\alpha op_u$. Let *State, Sem, QoS, R, Sem-QoS,* and *Sem-R* be sets of possible *state, semantically, QoS, R, semantically QoS,* and *semantically R-compatible* relations on methods of a class c, respectively. $\diamond_\alpha$ is symmetric. We assume that $\diamond_\alpha$ is transitive.

3. Compensating Methods

In traditional systems [1)], the state stored in the log is restored in the system. If the system is faulty, it is restarted. Multimedia objects are larger and more complex than simple objects such as tables. A method that removes the effect of another method is a *compensating* method [2)]. For example, suppose a method *paint* is performed on a *background* object. If *erase* is performed, the *background* object can be restored. The method *erase* is a compensating method for *paint*. Formally, a method op_u is a *compensating* method for another method op_t on a class c if $op_t \circ op_u(s) = s$ for every state s of the class c [2)].

[**Definition**] For every pair of methods op_t and op_u of a class c, op_u *α-compensates* for op_t ($op_u \rhd_\alpha op_t$) iff $(op_t \circ op_u) \Box_\alpha \phi$ for $\alpha \in \{$*State, Sem, R, Sem-R, QoS, Sem-QoS*$\}$. □

Let $(\sim_\alpha op)$ denote an *α-compensating* method for a method op with respect to the α-compensating relation, i.e., $op \circ (\sim_\alpha op) \Box_\alpha \phi$.

Let *State, Sem, QoS, R, Sem-QoS*, and *Sem-R* denote sets of possible *state, semantically, QoS, R, semantically QoS*, and *semantically R* compensating relations of methods of a class c. Let CR be a family of these compensating relations, $CR = \{\rhd_\alpha\}$.

Suppose $\alpha_1 \to \alpha_2$ for $\alpha_1, \alpha_2 \in CR$. For example, *Sem* $\to$ *Sem-R*. This means that op_t *Sem-r*-compensates op_u for RoS r in R ($op_t \rhd_{\equiv_r} op_u$) if $op_t \rhd_\equiv op_u$. Thus, $op_t \rhd_{\alpha_2} op_u$ if $op_t \rhd_{\alpha_1} op_u$.

[**Example 1**] Suppose a *movie* class is composed of the classes *car*, *words*, *music*, and *background*. *background* is furthermore composed of the classes *tree* and *cloud*. A *movie* state s_1 shows a colored video that includes all the components, as shown in **Fig. 1**. *background* and *car* in s_1 are removed by *del-car-bg* and then a state s_2 is obtained. Then, *monaural* is performed to obtain a monaural state s_3. Here, an application needs to undo the work done so far by *del-car-bg* and *monaural*. *stereo* is performed on s_3 and then a state s'_2 is obtained. *add-bg* is a method for adding a *background* object where *music* is *stereo*. A state

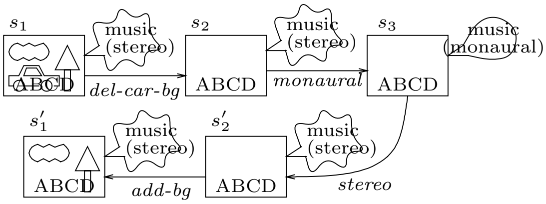


Fig. 1 Compensation.

s'_1 is obtained by performing *add-bg* on s'_2. If *car* is optional, $s'_1 \equiv s_1$, because all the other classes are the same as s_1. Hence, *add-bg* is a *Sem*-compensating method of *del-car-bg* (*add-bg* $\rhd_{\equiv}$ *del-bg-car*). $\Box$

After performing *op* on a state s of a class c, a state s' is obtained by performing the compensating method $(\sim_{Sem} op)$. $s' \equiv s$. From the theorem, *op* can be α_2-compensated for by $(\sim_{\alpha_1} op)$ instead of $(\sim_{\alpha_2} op)$ if $\alpha_1 \rightarrow \alpha_2$. For example, *add-bg* is $(\sim_{\equiv} \textit{del-car-bg})$ in Example 1. Suppose that *add-car-bg* is a method by which *car* and *background* objects are added. *add-car-bg* is $(\sim_{State} \textit{del-car-bg})$. A state obtained by performing *add-car-bg* is semantically equivalent to one obtained by performing *add-bg*. That is, if op' is $(\sim_{State} op)$, op' is $(\sim_{Sem} op)$.

[**Theorem**] $(\sim_{\alpha} op) \Box_{\beta} (\sim_{\beta} op)$ iff $\alpha \rightarrow \beta$. $\Box$

4. Classification of Methods

Suppose a method op_2 is performed after op_1, i.e., $op_1 \circ op_2$. Here, $op_1 \circ op_2$ is compensated for by a sequence of compensating methods $(\sim_{State} op_2) \circ (\sim_{State} op_1)$, i.e., $[op_1 \circ op_2 \circ (\sim_{State} op_2) \circ (\sim_{State} op_1)] - \phi$. For example, *erase* is $(\sim_{State} \textit{paint})$ and *degrade* is $(\sim_{State} \textit{upgrade})$. $(\sim_{State}(\textit{paint} \circ \textit{upgrade})) - [(\sim_{State} \textit{upgrade}) \circ (\sim_{State} \textit{paint})] - (\textit{degrade} \circ \textit{erase})$. Thus, the effect on the object o can be removed by performing the compensating methods for op_1 and op_2, 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* $(\sim_{\alpha}(op_1 \circ \ldots \circ op_n))$ is α_0-equivalent to 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 $\alpha, \alpha_0, \alpha_1, \ldots, \alpha_n \in \{$*State, Sem, QoS, R, Sem-QoS, Sem-R*$\}$. In this paper, we consider the case $\alpha_0 = \alpha$ for simplicity.

There are two types of methods: *state* methods to change the state of the object and *QoS* methods to change the QoS of the object. Similarly, there are two types of component classes,

Table 1 Types of methods.

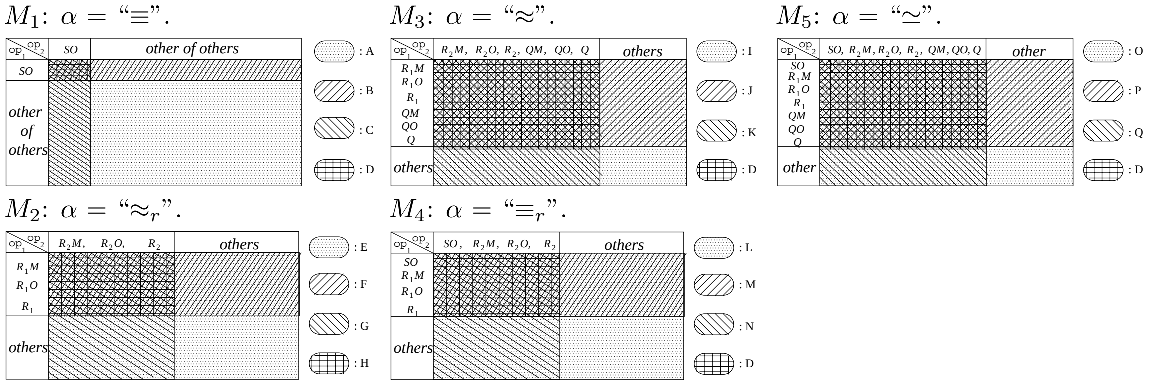
type	S/Q	M/O	condition
S	S		
SM	S	M	
SO	S	O	
Q	Q		
QM	Q	M	
QO	Q	O	
$R(r)$	Q		$Q(op_t(s)) \succeq r$.
RM (r)	Q	M	$Q(c_i(op_t(s))) \succeq r$ for every mandatory component class c_i of c.
RO (r)	Q	M	$Q(c_i(op_t(s))) \succeq r$ for every optional component class c_i of c.

S: state Q: QoS
M: mandatory O: optional

mandatory and optional ones, as discussed previously. Hence, there are *semantic* and *formal* types of methods, the former to change mandatory component objects and the latter to change optional objects but not mandatory ones. According to the properties, the methods are classified into the types shown in **Table 1**. Here, let $\tau(op)$ show a type of a method *op*, i.e., $\tau(op) \in \{S, SM, SO, Q, QM, QO, R, RM, RO\}$. Here, S and Q mean state and QoS methods, respectively. R shows a QoS method by which the QoS of an object is changed so that requirement QoS (RoS) is satisfied. M and O respectively indicate methods by which mandatory and optional components of an object are changed.

Let α_1 and α_2 be a pair of compensating relations of methods of a class c. We discuss how to compensate for $op_1 \circ op_2$, i.e., $(\sim_{\alpha}(op_1 \circ op_2)) \Box_{\alpha} [(\sim_{\alpha_2} op_2) \circ (\sim_{\alpha_1} op_1)]$ holds on the basis of method types $\tau(op_1)$ and $\tau(op_2)$ for $\alpha, \alpha_1, \alpha_2 \in \{State, Sem, QoS, R, Sem\text{-}QoS, Sem\text{-}R\}$. In **Fig. 2**, each entry $M_i(\tau_1, \tau_2)$ shows a condition for which $(\sim_{\alpha}(op_1 \circ op_2)) \Box_{\alpha} [(\sim_{\alpha_2} op_2) \circ (\sim_{\alpha_1} op_1)]$ holds for types τ_1 and τ_2 of methods op_1 and op_2 $(i = 1, \ldots, 5)$. In the matrices, $\alpha_j = \phi$ shows "$(\sim_{\alpha_j} op_j)$ is not performed". For example, if $\tau(op_1) = SO$ and $\tau(op_2) = S$, $M_1(SO, S) =$ B, i.e., $(\sim_{Sem}(op_1 \circ op_2)) \equiv (\sim_{State} op_2)$. Since objects are manipulated by op_1, $op_1(s) \equiv s$ for every state s, i.e., $(\sim_{\alpha} op_1)$ does not need to be performed.

Table 2 summarizes what types of QoS relations, α_1, α_2, and α satisfy the compensation $[(\sim_{\alpha_2} op_2) \circ (\sim_{\alpha_1} op_1)] \rhd_{\alpha} (op_1 \circ op_2)$. Here, "$\alpha =$ - " means any one of $\{State, Sem, QoS, R, Sem\text{-}QoS, Sem\text{-}R\}$ and "α" of α_i means "$\alpha_i = \alpha$". For example, $(\sim_{\alpha}(op_1 \circ op_2)) - [(\sim_{State} op_1) \circ (\sim_{State} op_2)]$. This means that $op_1 \circ op_2$ can be



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: $\alpha_1 = \alpha_2 = \phi$.
E: $\alpha_1, \alpha_2 \in \{State, r\}$.
F: $\alpha_1 = \phi \ \wedge \ \alpha_2 \in \{State, r\} \ \wedge \ r_2 \cap Q(op_1(s)) \succeq r$.
G: $\alpha_1 \in \{State, r\} \ \wedge \ \alpha_2 = \phi \ \wedge \ r_1 \cap Q(op_2(s)) \succeq r$.
H: $\alpha_1 = \alpha_2 = \phi \ \wedge \ r_1 \cap r_2 \succeq r$.
I: $\alpha_1 = \alpha_2 \in \{State, QoS, r\}$.
J: $\alpha_1 = \phi \ \wedge \ \alpha_2 \in \{State, QoS, r\}$.
K: $\alpha_1 \in \{State, QoS, r\} \ \wedge \ \alpha_2 = \phi$.
L: $\alpha_1, \alpha_2 \in \{State, Sem, r, Sem\text{-}r\} \ \wedge \ r_1 \cap r_2 \succeq r$.
M: $\alpha_2 \in \{State, Sem, r, Sem\text{-}r\} \ \wedge \ \alpha_1 = \phi \ \wedge \ r_2 \cap Q(op_1(s)) \succeq r$.
N: $\alpha_1 \in \{State, Sem, r, Sem\text{-}r\} \ \wedge \ \alpha_2 = \phi \ \wedge \ r_1 \cap Q(op_2(s)) \succeq r$.
O: $\alpha_1, \alpha_2 \in \{State, Sem, QoS, R, Sem\text{-}QoS, Sem\text{-}R\}$.
P: $\alpha_2 \in \{State, Sem, QoS, R, Sem\text{-}QoS, Sem\text{-}R\} \ \wedge \ \alpha_1 = \phi$.
Q: $\alpha_1 \in \{State, Sem, QoS, R, Sem\text{-}QoS, Sem\text{-}R\} \ \wedge \ \alpha_2 = \phi$.

Fig. 2 Conditions.

Table 2 Compensation.

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

compensated for by $(\sim_{state} op_1) \circ (\sim_{state} op_2)$ for every requirement α.

[**Theorem**] An α-equivalent relation "$(\sim_\alpha (op_1 \circ op_2)) \square_\alpha [(\sim_{\alpha_2} op_2) \circ (\sim_{\alpha_1} op_1)]$" holds iff one of the relations shown in Table 2 holds. □

5. Reduced Compensating Sequence

If op_1 is *State-compatible* with op_2, $(op_1 \diamond_{State} op_2)$, $(op_1 \circ op_2) - (op_2 \circ op_1)$. Hence, $op_1 \circ op_2$ can also be compensated for by $(\sim_{State} op_1) \circ (\sim_{State} op_2)$ while compensated for by $(\sim_{State} op_2) \circ (\sim_{State} op_1)$. $[(\sim_{State} op_1) \circ (\sim_{State} op_2)] - [(\sim_{State} op_2) \circ (\sim_{State} op_1)]$. Thus, if a pair of methods are α-compatible, they can be exchanged in a sequence. For a pair of methods op_1 and op_2, $op_1 \diamond_\alpha op_2$ iff $(\sim_\alpha op_1) \diamond_\alpha (\sim_\alpha op_2)$. By using this α-compatibility relation, the computation order of the methods can be changed. Let S be a sequence $S_1 \circ op_1 \circ S_2 \circ op_2 \circ S_3$ where S_1, S_2, and S_3 are subsequences of methods. Let S' be another sequence $S_1 \circ op_2 \circ S_2 \circ op_1 \circ S_3$. Here, $S \square_\alpha S'$ (S is α-equivalent with S') if $op_1 \diamond_\alpha op_2$, $op \diamond_\alpha op_1$, and $op \diamond_\alpha op_2$ for every method op in S_2. It is straightforward to show that "$(\sim_\alpha (S_1 \circ op_1 \circ S_2 \circ op_2 \circ S_3)) \square_\alpha [(\sim_\alpha S_3) \circ (\sim_\alpha op_1) \circ (\sim_\alpha S_2) \circ (\sim_\alpha op_2) \circ (\sim_\alpha S_1)]$" holds.

add $\diamond_r$ *grayscale*, where r denotes that "the application does not require colors". Suppose *add* is performed before *grayscale*, i.e., *add* ∘ *grayscale*. This sequence is *r-compensated* for by $(\sim_r grayscale) \circ (\sim_r add)$. However, it takes a shorter time to perform $(\sim_r grayscale)$ after removing a car which is added by *add*, i.e., $(\sim_r add)$, because the number of objects whose colors are to be changed is decreased. Hence, *add* ∘ *grayscale* can be more efficiently compensated by $(\sim_r add) \circ (\sim_r grayscale)$.

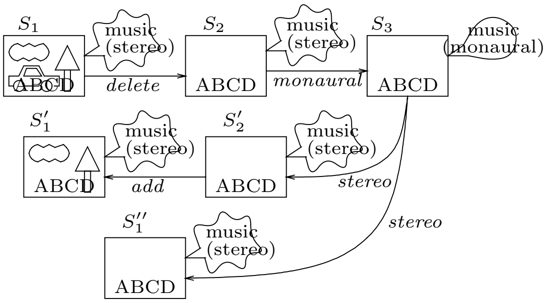


Fig. 3 Compensating for a sequence of methods.

Next, let us consider how to reduce the number of compensating methods to compensate for a sequence of methods. Suppose a *car* object c is deleted after being added, i.e., $add \circ delete$. Since $(add \circ delete) - \phi$ holds, $(\sim_{State} delete) \circ (\sim_{State} add)$ is not required to be performed. Next, suppose a method $paint_1$ that paints an object *red* is performed after the object has been painted *yellow* by $paint_2$. $paint_2 \circ paint_1$ brings the same result obtained by performing only $paint_1$, i.e., $(paint_2 \circ paint_1) - paint_1$. In order to compensate for $paint_1 \circ paint_2$, only $(\sim_\alpha paint_1)$ can be performed. op_t is an *α-identity* method iff $op_t \,\Box_\alpha\, \phi$. op_t *α-absorbs* op_u iff $(op_u \circ op_t) \,\Box_\alpha\, op_t$.

[**Example 2**] Let us consider a *karaoke* object k shown in **Fig. 3**. A state s_3 of k is obtained by performing $delete \circ monaural$ on a state s_1. *stereo* is a *State*-compensating method for *monaural*. Hence, $(\sim_{State}(delete \circ monaural)) - (stereo \circ add)$. In the *karaoke* object k, *background* and *car* objects are optional. A state s_1'' is obtained by performing *stereo* on the state s_3. s_1'' is semantically equivalent to s_1 ($s_1'' \equiv s_1$). An application considers s_1'' to be the same as s_1. Hence, $delete \circ monaural$ can be undone by performing one method *stereo*. $(\sim_{\equiv}(delete \circ monaural)) \equiv stereo$. □

Next, we discuss how to reduce a sequence of methods. Let S be a sequence $S_1 \circ S_2 \circ S_3$ where S_1, S_2, and S_3 are subsequences of methods. If S_2 is an *α-identity* sequence, $(\sim_\alpha(S_1 \circ S_2 \circ S_3)) \,\Box_\alpha\, (\sim_\alpha(S_1 \circ S_3))$. If S_3 *α-absorbs* S_2, $(\sim_\alpha(S_1 \circ S_2 \circ S_3)) \,\Box_\alpha\, (\sim_\alpha(S_1 \circ S_3))$. If S_2 is α-compatible with S_3 ($S_2 \diamond_\alpha S_3$), $(\sim_\alpha(S_1 \circ S_2 \circ S_3)) \,\Box_\alpha\, (\sim_\alpha(S_1 \circ S_3 \circ S_2))$.

Let S be a sequence of methods performed on an object o. S is partitioned into a sequence of subsequences $S_1 \circ \ldots \circ S_m (m \geq 1)$. For every $S_i = \alpha_{i1} \circ \ldots \circ \alpha_{il_i}$, every pair of methods in S_i are α-compatible. In addition, every method op_{ij} α-conflicts with $op_{i-1,l_{i-1}}$ in S_{i-1} and $op_{i+1,l_{i+1}}$ in S_{i+1}. Each subsequence S_i is reduced though the following **Reduce** by using the α-identity and α-absorbing relations.

Let S be a sequence of methods performed on an object o that are to be α-compensated for. Let S_1 and S_2 be compensating sequences for S, i.e., $(S \circ S_1) \,\Box_\alpha\, \phi$ and $(S \circ S_2) \,\Box_\alpha\, \phi$. If it takes a shorter time to perform S_1 than S_2 and S_1 consumes a smaller amount of computation resources than S_2, S_1 is *cheaper* than S_2. Since it is not easy to define the *cost*, S_1 is defined to be *cheaper* than S_2 if $|S_1| \leq |S_2|$. Here, $|S_i|$ denotes the number of methods in a sequence S_i. A cheaper sequence S' is found for a sequence S by the following procedure:

1. Let S be a sequence $S'' \circ op$.
2. $S' =$ **Reduce**(S'', op).

Reduce(S', op).

1. If $S' = \phi$, $S_1 := op$; **return** (S_1);
2. Let S' be $S'' \circ op'$.
3. If op α-absorbs op', op' is removed from S', i.e., $S' := S''$ and $S_1 :=$ **Reduce**(S'', op); **return** (S_1);
4. If $op \diamond_\alpha op'$, $S_1 :=$ **Reduce**($S'' \circ op$, op'); $S_2 :=$ **Reduce**(S'', op')$\circ op$ if $|S_1| < |S_2|$, **return** (S_1) else **return** (S_2).
5. else $S_1 :=$ **Reduce**(S'', op') $\circ$ op, **return** (S_1);

Let $|op|$ be a number of methods to be performed. Here, $|op| = 1$ and $|S \circ op| = |S| + 1$. In Fig. 3, **Reduce**$(\sim_{\equiv}(delete \circ monaural)) = stereo$ since $|stereo \circ add| \geq |stereo|$.

6. Concluding Remarks

In multimedia systems, the QoS of an object is manipulated in addition to the state of the object. In this paper, we have discussed how the QoS of the object is manipulated by methods. We defined semantically, QoS, RoS, semantically QoS, and semantically RoS conflicting relations among methods of multimedia objects. By using the relations, we defined compensating methods to undo the work done by the methods. We also made clear how types of compensating methods are related from the QoS point of view. We discussed how to construct a compensating sequence of methods that implies better performance.

References

1) Bernstein, P.A., Hadzilacos, V. and Goodman, N.: *Concurrency Control and Recovery in Database Systems*, Addison-Wesley Publishing

Company (1987).
2) Korth, H.F., Levy, E. and Silberschalz, A.: A Formal Approach to Recovery by Compensating Transactions, *Proc. VLDB*, pp.95–106 (1990).
3) MPEG Requirements Group: MPEG-4 Requirements, ISO/IEC JTC1/SC29/WG11, N2321 (1998).
4) Stroustrup, B.: *The C++ Programming Language* (2nd ed.), Addison-Wesley (1991).
5) Nemoto, N., Tanaka, K. and Takizawa, M.: QoS-based Synchronization of Multimedia Objects, *Proc. 11th Int'l Conf. on Database and Expert Systems Applications* (*DEXA '00*), pp.151–160 (2000).
6) Yokoyama, M., Tanaka, K. and Takizawa, M.: QoS-Based Recovery of Multimedia Objects, *Proc. IEEE Int'l Conf. on Parallel and Distributed Systems (ICPADS-00) Workshops*, pp.43–48 (2000).
7) Yokoyama, M., Nemoto, N., Tanaka, K. and Takizawa, M.: Quality-Based Approach to Manipulating Multimedia Objects, *Proc.2000 Int'l Conf. on Information Society in the 21st Century: Emerging Technologies and New Challenges (IS2000)*, pp.380–387 (2000).

(Received June 7, 2001)
(Accepted November 14, 2001)

Motokazu Yokoyama was born in 1976. He received his B.E. degree in Computers and Systems Engineering from Tokyo Denki Univ., Japan, in 2000. He is now a graduate student of the master course in the Dept. of Computers and Systems Engineering, Tokyo Denki Univ.. His research interests include distributed multimedia networks. He is a member of IPSJ.

Katsuya Tanaka was born in 1971. He received his B.E. and M.E. degree in Computers and Systems Engineering from Tokyo Denki Univ., Japan in 1995 and 1997, respectively. From 1997 to 1999, he worked for NTT Data Corporation. Currently, he is a research associate in the Dept. of Computers and Systems Engineering, Tokyo Denki University. He received the D.E. degree from Dept. of Computers and Systems Engineering, Tokyo Denki University, Japan, in 2000. His research interests include distributed systems, transaction management, recovery protocols, and computer network protocols. He is a member of IEEE CS and IPSJ.

Makoto Takizawa was born in 1950. He received his B.E. and M.E. degrees in Applied Physics from Tohoku Univ., Japan, in 1973 and 1975, respectively. He received his D.E. in Computer Science from Tohoku Univ. in 1983. From 1975 to 1986, he worked for Japan Information Processing Developing Center (JIPDEC) supported by the MITI. He is currently a Professor of the Dept. of Computers and Systems Engineering, Tokyo Denki Univ. since 1986. He is also now a Dean of the graduate school of Science and Engineering, Tokyo Denki Univ. From 1989 to 1990, he was a visiting professor of the GMD-IPSI, Germany. He is also a regular visiting professor of Keele Univ., England since 1990. He is now a general co-chair of IEEE ICDCS-2002, Vienna. He was a program co-chair of IEEE ICDCS-18, 1998 and serves on the program committees of many international conferences. He chaired SIGDPS of IPSJ from 1997 to 1999. He is IPSJ fellow. His research interests include communication protocols, group communication, distributed database systems, transaction management, and security. He is a member of IEEE, ACM, and IPSJ.