Optimization of Compensation in Object-Based Multimedia Systems

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

In distributed applications, QoS of a multimedia object is manipulated in addition to the state. While objects are manipulated through methods, the manipulations on the objects have to be undone in designing multimedia systems and recovering from the system fault. In this paper, we discuss how methods performed are compensated by other methods. Novel types of compensating methods are defined to obtain a state and QoS of the object which satisfy requirements. We discuss how to find a cheaper way to compensate a sequence of methods.

1 Introduction

Distributed applications are composed of multimedia objects. Here, quality of service (QoS) of a multimedia object is manipulated as well as the state. The authors [7,8] define novel types of conflicting relations among methods with respect to QoS while the traditional definition is based on states of objects. The paper [8] discusses novel types of serializability based on QoS and concurrency control mechanism on multimedia objects.

In manipulating a multimedia object, an application might like to undo the manipulation, for example, for interactively designing and implementing an application. In another example, an object is rolled back due to the fault of the object. Suppose that an application changes a colored *movie* object to a monochrome one by a method *grayscale* after adding a *red* car by a method *add-car*. Here, the *movie* object is monochrome. Next, suppose the application would like to undo the manipulation done here. According to the traditional ways, the *movie* object is rolled back to the previous one saved at a checkpoint [2], i.e. colored object without the *car* object. Another way is to compensate a computation sequence of *add-car* and *grayscale* by other methods. *del-car* is a method where a *car* is removed. *color* is a method where a *scene* object is changed to be colored. If *color* is performed after *del-car*, the object is recovered to the previous state. Here, *del-car* and *color* are referred to as *compensating* methods of *add-car* and *grayscale*, respectively. If the application is not interested in how colorful the *movie* object is, only the *car* object can be removed without changing the color. That is, the sequence of methods *add-car* and *grayscale* can be just compensated by one method *del-car* with respect to QoS required by the application. In the paper [9], the authors discussed a novel way to compensate methods performed on a multimedia object where QoS and the state of the object are changed so as to satisfy the user's requirements. A sequence of *add-car* and *grayscale* is compensated by a sequence of compensating methods *color* and *del-car* as presented here. Here, the previous state can be also obtained by performing *color* after *del-car*. The latter compensating sequence is cheaper than the former because the *car* object is not colored in the latter one. We discuss how to find a better sequence of *compensating* methods.

In section 2, we discuss relations among methods. In section 3, we discuss compensating methods. In section 4, we discuss how to compensate a sequence of methods.

2 QoS-based Relations of Methods

An object-based system is composed of *classes* and *objects* [6]. 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 subclass c_i. A state of an object is changed by performing a method op. 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. "$op_1 \circ op_2$" shows a serial computation of op_1 and op_2.

Applications obtain service of an object o through methods. Each service is characterized by *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 parameter like frame rate. A QoS *value* q_1 *dominates* another QoS value q_2 ($q_1 \succeq q_2$) iff

q_1 shows a better level of QoS than q_2. For example, $\langle 160 \times 120[\text{pixels}], 1024[\text{colors}], 15[\text{fps}]\rangle \succeq \langle 120 \times 100, 512, 15\rangle$. $q_1 \cup q_2$ and $q_1 \cap q_2$ show least upper bound and greatest lower bound of q_1 and q_2 on $\succeq$, respectively. Let $Q(s)$ be a QoS value of a state s of an object o. $Q(op(s))$ and $Q([op(s)])$ are QoS values of state and output obtained by performing *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 either *mandatory* or *optional*. There are the following relations among a pair of states s_t and s_u of a class c [7,9]:

- s_t is *state-consistent* with s_u $(s_t - s_u)$ iff $s_t = s_u$.
- s_t is *semantically 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 mandatory component class c_i of c.
- s_t is *QoS-consistent* 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-consistent* with s_u $(s_t \simeq s_u)$ iff $s_t \approx s_u$ or $c(s_t) \simeq c(s_u)$ for every mandatory component class c_i of c.
- s_t is *r-consistent* with 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-consistent* with 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 *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 a same car with different QoS and t_1 and t_2 indicate a same tree with different QoS.

Let $\Box_\alpha$ show an α-consistent relation where α shows some consistent relation. For example, $\Box_{QoS}$ (or $\Box_\approx$) shows "$\approx$". *State, Sem, QoS, R, Sem-QoS,* and *Sem-R* stand for sets of possible *state, semantically, QoS, R, semantically QoS*, and *semantically R* consistent 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 value$\}$. Let C be a family of the sets *state, Sem, QoS, R, Sem-QoS*, and *Sem-R* of consistent relations. A relation "$a \rightarrow b$" for a pair of sets a and b shows 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. $State \rightarrow Sem, State \rightarrow R, R \rightarrow Sem\text{-}R$ $R \rightarrow QoS, QoS \rightarrow Sem\text{-}QoS, Sem\text{-}R \rightarrow Sem\text{-}QoS$ are primitive relations, i.e. not transitive.

Let op_t and op_u be a pair of methods 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 the class 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, $\Box_{r_1} \rightarrow \Box_{r_2}$ 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 [1,4], 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 $\alpha \in C$. □

For example, a method op_t is *semantically compatible* with a method op_u $(op_t \mid\mid\mid op_u)$ iff $(op_t \circ op_u) \equiv (op_u \circ op_t)$. The "*R-compatible* relation" $\Diamond_R$ shows a set $\{ \Diamond_r \mid r \in R\}$ of consistent relations on various RoS 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 and transitive.

3 Compensating Methods

3.1 Compensation

In traditional systems [1], if the system is faulty, the state stored in the log is restored in the system and then the system is restarted. Suppose *paint* is performed on a *background* object. If *erase* is performed, the *background* object can be restored. *erase* is a compensating method of *paint*. Traditionally, 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 [4]. We extend the compensation concept to multimedia objects.

[Definition] A method op_u *α-compensates* another method op_t on an object $(op_u \rhd_\alpha op_t)$ with respect to a consistent relation α in C iff $(op_t \circ op_u) \Box_\alpha \phi$. □

Let $(\sim_\alpha op)$ denote an *α-compensating* method of a method op, 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, $CR = \{\rhd_\alpha \mid \alpha \in C\}$.

Suppose $\alpha_1 \rightarrow \alpha_2$ for $\alpha_1, \alpha_2 \in CR$. For example, *Sem* $\rightarrow$ *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 $op_t \rhd_{\equiv} op_u$.

[Theorem] If $\alpha_1 \rightarrow \alpha_2$, $op_t \rhd_{\alpha_2} op_u$ if $op_t \rhd_{\alpha_1} op_u$. $\Box$

[Example 1] Suppose a *movie* class is composed of classes *car*, *words*, *music*, and *background*. The class *background* is furthermore composed of classes *tree* and *road*. A *movie* state s_1 shows a colored video which includes all the components as shown in Figure 1. Objects *background* and *car* in s_1 are removed by performing a method *del-car-bg* and then a state s_2 is obtained. Then, *monoral* is performed to obtain a monoral state s_3. Here, an application would like to undo the work done so far by methods *delete* and *monoral*. *stereo* is performed on s_3 and then a state s_2' is obtained. *add-bg* is a method to add a *background* object where *music* is *stereo*. A state s_1' is obtained by performing a method *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, a method *add-bg* is a *Sem*-compensating method of a method *del-car-bg* (*add-bg* $\rhd_{\equiv}$ *del-bg-car*). $\Box$

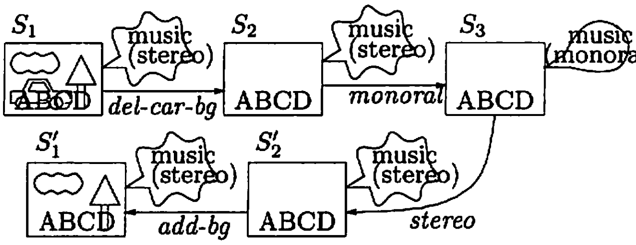


Figure 1: Compensation.

After performing a method *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, the method *op* can be α_2-compensated by ($\sim_{\alpha_1} op$) instead of ($\sim_{\alpha_2} op$) if $\alpha_1 \rightarrow \alpha_2$. For example, a method *add-bg* is ($\sim_{\equiv}$ *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 ($\sim_{state}$ *del-car-bg*). A state obtained by performing *add-car-bg* is semantically consistent with one obtained by performing *add-bg*.

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

3.2 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 by a sequence of *state*-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 ($\sim_{State}$*paint*) and *degrade* is ($\sim_{State}$*upgrade*). $\sim_{State}$(*paint* $\circ$ *upgrade*) $-$ ($\sim_{State}$*upgrade*)$\circ$($\sim_{State}$*paint*)$-$(*degrade*$\circ$*erase*). Thus, the effect on the object o can be removed by performing the compensating methods of op_1 and op_2, i.e. $\sim_{State}(op_1 \circ op_2) - (\sim_{State} op_2) \circ (\sim_{State} op_1)$. 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-consistent with a sequence of compensating methods $(\sim_{\alpha_n} op_n) \circ \ldots \circ (\sim_{\alpha_1} op_1)$.

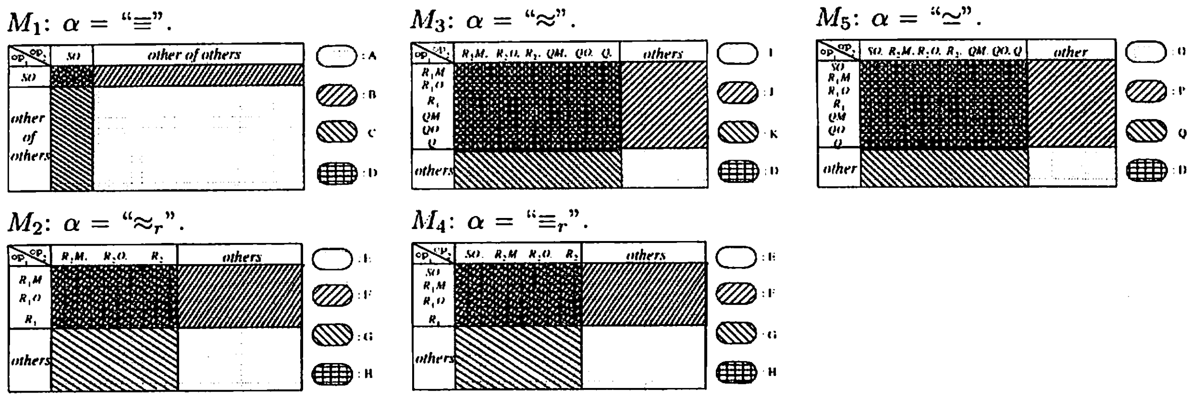
[Problem] Find consistent relations $\alpha_0, \alpha_1, \ldots, \alpha_n$ for α such that $\sim_\alpha(op_1 \circ \ldots \circ op_n) \Box_{\alpha_0} (\sim_{\alpha_n} op_n) \circ \ldots \circ (\sim_{\alpha_1} op_1)$. $\Box$

In this paper, we consider a case $\alpha_0 = \alpha$ for simplicity.

There are two types of methods, *state* one to change the state of the object and *QoS* one to change QoS of the object. For example, *add-car* is a state method and *grayscale* is a QoS method. There are two types of component classes, mandatory and optional ones as discussed before. Hence, there are *semantical* and *formal* types of methods, the first one to change the mandatory component object and the other one to change optional component object but not mandatory ones. The methods are classified into types shown in Table 1. S and Q mean state and QoS methods, respectively. R shows a QoS method by which QoS of an object is changed so that RoS is satisfied. M and O indicate methods by which mandatory and optional components of an object are changed, respectively. Let T show a set $\{S, SM, SO, Q, QM, QO, R, RM, RO\}$. Here, let $\tau(op)$ show a type of a method op, i.e. $\tau(op) \in T$.

Let α, α_1, and α_2 be consistent relations in C for a class c. We discuss how to compensate a sequence $op_1 \circ op_2$, 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 $\tau_1 = (op_1)$ and $\tau_2 = (op_2)$. In Figure 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 matrixes, $\alpha_j = \phi$ 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. $(\sim_\alpha op_1)$ is not required to be performed.

Table 2 summarizes what types of consistent 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 in C and "α" of α_i means "$\alpha_i = \alpha$". For example, $\sim_\alpha(op_1 \circ op_2) -$



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: $\alpha_1 \in \{State, Sem, QoS, R, Sem\text{-}QoS, Sem\text{-}R\} \ \wedge \ \alpha_2 = \phi$.

Figure 2: Conditions.

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		$op_t(s) \succeq r$.
RM (r)	Q	M	$c_i(op_t(s)) \succeq r$ for every mandatory component class c_i of c.
RO (r)	Q	M	$c_i(op_t(s)) \succeq r$ for every optional component class c_i of c.

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

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*

$\{(\sim_{State} op_1) \circ (\sim_{State} op_2)\}$. This means, $op_1 \circ op_2$ can be compensated by $(\sim_{state} op_1) \circ (\sim_{state} op_2)$ for every requirement α.

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

4 Reduced Compensating Sequence

4.1 Compensating sequence

Suppose a *background* object b is manipulated by a method *grayscale* after *add-car* as presented before. Here, a colored object b with a *red* car is changed to a monochromatic state b'. b' can be recovered to the previous state b by performing compensating methods *color* ∘ *del-car*. Here, *color* and *del-car* are *State*-compensating methods of *grayscale* and *add-car*, i.e. ($\sim_{state}$*grayscale*) and ($\sim_{state}$*add-car*), respectively. b' can be also recovered to b by performing *del-car* ∘ *color* because *del-car* and *color* are State-compatible. Thus, *add-car* ∘ *grayscale* can be compensated by any of (*color* ∘ *del-car*) and (*del-car* ∘ *color*). We discuss how to take a cheaper compensating sequence.

If a method op_1 is *State-compatible* with a method op_2 ($op_1 \mid op_2$), $(op_1 \circ op_2) - (op_2 \circ op_1)$. Hence, $op_1 \circ op_2$ can be also compensated by $(\sim_{State} op_1) \circ (\sim_{State} op_2)$ while compensated 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 with respect to consistent relation α in C, the methods can be exchanged in a sequence. A method op_1 is α-compatible with a method 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 methods can be changed. Let S be a sequence $op_1 \circ S_1 \circ op_2$ of methods where S_1 is a subsequence of methods and op_1 and op_2 are methods. Let S' be another sequence $op_2 \circ S_1 \circ op_1$. Here, $S \sqcap_\alpha S'$ (S is α-consistent 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_1. This means op_1 and op_2 can be exchanged in the sequences. Here, it is straightforward "$\sim_\alpha(op_1 \circ S_1 \circ op_2) \sqcap_\alpha (\sim_\alpha op_1) \circ (\sim_\alpha S_1) \circ (\sim_\alpha op_2)$" holds.

Let r show RoS "application is not interested in colors". A method *add-car* is r-compatible with a method *grayscale* ($add - car \diamond_r grayscale$). Suppose *add-car* is performed before *grayscale*, i.e. $add - car \circ grayscale$. This sequence is *r-compensated* by $(\sim_r grayscale) \circ (\sim_r add - car)$. However, it takes a shorter time to perform $(\sim_r grayscale)$ after removing a car which is added by *add-car*, i.e. $(\sim_r add - car)$, because the number of objects whose colors to be changed are decreased. Hence, $add - car \circ grayscale$ can be more efficiently compensated by $(\sim_r add - car) \circ (\sim_r grayscale)$ with respect to RoS r. The method *del-car* is an r-compensating method of *add-car*, i.e. *del-car* = ($\sim_r$*add-car*) = ($\sim_{state}$*add-car*). Since the application is not interested in color, $(\sim_r grayscale)$ can be omitted, i.e. ϕ is $(\sim_r grayscale)$.

4.2 Optimization

Next, let us consider how to reduce the number of compensating methods to compensate a sequence of methods. Suppose a *car* object c is deleted after added, i.e. *add-car* ∘ *del-car*. Since (*add-car* ∘ *del-car*) $-\phi$ holds, ($\sim_{State}$*del-car*) ∘ ($\sim_{State}$*add-car*) is not required to be performed. Next, suppose a method $paint_1$ which paints an object *red* is performed after painting *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 $paint_1 \circ paint_2$, only $(\sim_\alpha paint_1)$ can be performed. The following relations are defined for methods op_t and op_u and a consistent relation α:

- op_t is an α-*identity* method iff $op_t \sqcap_\alpha \phi$.
- op_t α-*absorbs* op_u iff $(op_t \circ op_u) \sqcap_\alpha op_t$.

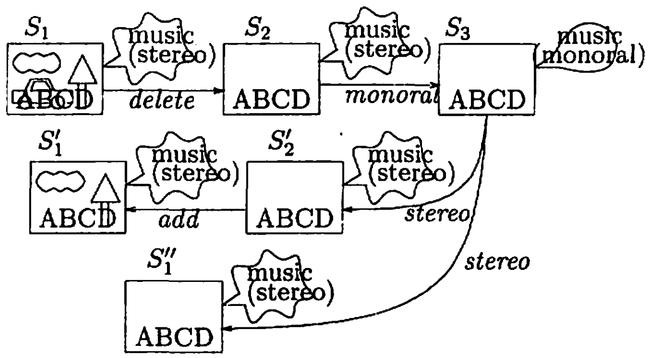


Figure 3: Compensating sequence of methods.

[Example 2] Let us consider a *karaoke* object k shown in Figure 3. A state s_3 of the *karaoke* object k is obtained by performing a sequence of methods *del-car-bg* ∘ *monoral* on a state s_1. A method *stereo* is a *State*-compensating method of *monoral*. Hence, $\sim_{State}$(*del-car-bg* ∘ *monoral*) − (*stereo* ∘ *add-bg*). In the *karaoke* object k, *background* and *car* objects are optional. A state s_1'' is obtained by performing the method *stereo* on the state s_3. The state s_1'' is semantically consistent with the state s_1 ($s_1'' \equiv s_1$). That is, an application considers the state s_1'' to be the same as the state s_1. Hence, the method sequence *del-car-bg* ∘ *monoral* can be undone by performing one method *stereo*. $\sim_\equiv$(*del-car-bg* ∘ *monoral*) ≡ *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, $\sim_\alpha(S_1 \circ S_2 \circ S_3) \sqcap_\alpha \sim_\alpha(S_1 \circ S_3)$. If S_3 α-*absorbs* S_2, $\sim_\alpha(S_1 \circ S_2 \circ S_3) \sqcap_\alpha \sim_\alpha(S_1 \circ S_3)$. If S_2 is α-compatible with S_3 ($S_2 \diamond_\alpha S_3$), $\sim_\alpha(S_1 \circ S_2 \circ S_3) \sqcap_\alpha \sim_\alpha(S_1 \circ S_3 \circ S_2)$.

Let S be a sequence of methods performed on an object o. S is partitioned into a sequence of subsequences $S_1 \circ \ldots \circ S_m (m \geq 1)$. The subsequences satisfy the following conditions:

1. For every subsequence $S_i = op_{i1} \circ \ldots \circ op_{il_i}$, every pair of methods op_{ij} and op_{ik} in S_i are α-compatible.

2. Every method op_{ij} in S_i α-conflicts with methods $op_{i-1,l_{i-1}}$ in S_{i-1} and $op_{i+1,l_{i+1}}$ in S_{i+1}.

A subsequence which satisfies the conditions presented above is referred to as *segment*.

We take a following strategy.

1. A sequence S of methods is partitioned into segments $S_1, \ldots, S_m$.
2. Each segment S_i is reduced into a subsequence S'_i.

Each subsequence S_i is reduced though the following procedure **Reduce** by using the α-identity and α-absorbing relations.

Let S be a sequence of methods performed on an object o are to be α-compensated. Let S_1 and S_2 be compensating sequences of 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 less amount of computation resource 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 $S'' \circ op$ where S'' is a subsequence and op is a method.
2. $S' = \textbf{Reduce}(S'', op)$.

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 $S_1 := \textbf{Reduce}(S'', op)$; **return** (S_1);
4. If $op \Diamond_\alpha op'$, $S_1 := \textbf{Reduce}(S'' \circ op, op')$; $S_2 := \textbf{Reduce}(S'', op') \circ op$ of $|S_1| < |S_2|$, **return** (S_1) else **return** (S_2).
5. else $S_1 := \textbf{Reduce}(S'', op') \circ op$, **return** (S_1);

Let $|S|$ be a number of methods to be performed in a sequence S. $|S|$ is defined as follows: $|op| = 1$ and $|S \circ op| = |S| + 1$. In Figure 3, $\textbf{Reduce}(\sim_\equiv(\textit{delete} \circ \textit{monoral})) = \textit{stereo}$ since $|\textit{stereo} \circ \textit{add}| \geq |\textit{stereo}|$.

5 Concluding Remarks

In multimedia systems, QoS of an object is manipulated in addition to the state of the object. In this paper, we 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 works 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 which imply 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] 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.

[3] 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*, 2000, pp.43–48.

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

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

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

[7] Nemoto, N., Tanaka, K., and Takizawa, M., "QoS-based Synchronization of Multimedia Objects," *Proc. of the 11th Int'l Conf. on Database and Expert Systems Applications (DEXA'00)*, 2000, pp.151–160.

[8] Nemoto N., Tanaka K., and Takizawa M., "Quality-Based Synchronization Methods of Multimedia Objects," to appear in *Information Sciences an International Journal*, 2001.

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