Compensation of Methods for Multimedia Applications *

1G－05

Motokazu Yokoyama, Katsuya Tanaka, and Makoto Takizawa †
Tokyo Denki University ‡
Email : {moto, katsu, taki}@takilab.k.dendai.ac.jp

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.

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, 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 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*. 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 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:

- 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 "$\approx$".

In the traditional theories, 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, op_t is *semantically compatible* with op_u $(op_t \, ||| \, op_u)$ iff $(op_t \circ op_u) \equiv (op_u \circ op_t)$. The "*R-compatible* relation" $\Diamond_R$ shows a set $\{ \Diamond_r | r \in R\}$ 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 and transitive.

3 Compensating Methods

In traditional systems, if the system is faulty, the state stored in the log is restored in the system and then

*マルチメディアアプリケーションにおける補償演算方式
†横山 基一, 田中 勝也, 滝沢 誠
‡東京電機大学

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. 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 $(op_t \circ op_u) \Box_\alpha \phi$. □

4 Reduced Compensating Sequence

Let r show RoS "application is not interested in colors". A method *add-car* is r-compatible with *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* = $(\sim_r add\text{-}car) = (\sim_{state} add\text{-}car)$. Since the application is not interested in color, $(\sim_r grayscale)$ can be omitted, i.e. ϕ is $(\sim_r grayscale)$.

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, $(\sim_{State} del\text{-}car) \circ (\sim_{State} add\text{-}car)$ 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 $op_t \Box_\alpha \phi$.
- op_t *α-absorbs* op_u iff $(op_t \circ op_u) \Box_\alpha op_t$.

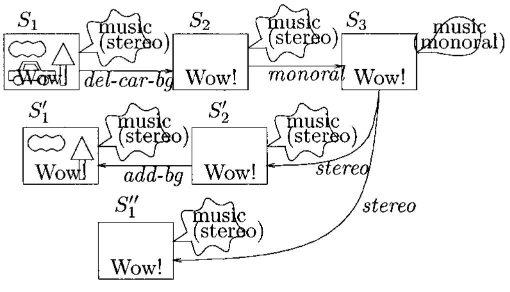

Figure 1: Compensating sequence of methods.

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 $S_1 \circ \ldots \circ S_m (m \geq 1)$. 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' = **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 of $|S_1| < |S_2|$, **return** (S_1) else **return** (S_2).
5. else $S_1 :=$ **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 1, **Reduce**($\sim_\equiv$(*delete* ∘ *monoral*)) = *stereo* since $|stereo \circ add| \geq |stereo|$.

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] Yokoyama, M., Tanaka, K., and Takizawa, M., "QoS-Based Method for Compensating Multimedia Objects," *Proc. of DEXA Int'l Workshop on Network-Based Information Systems (NBIS-4)*, 2001, pp.185–189.