Vol. 40 No. 1 Transactions of Information Processing Society of Japan Jan. 1999

*Regular Paper*

# Quality-based Flexibility in Distributed Systems

TETSUO KANEZUKA,[†] HIROAKI HIGAKI[†] and MAKOTO TAKIZAWA[†]

This paper discusses how to make a distributed multimedia object system flexible so as to satisfy applications' requirements in change of the system environment. The system change is modeled to be the change of not only for types of service but also quality of service (QoS) supported by the objects. A method changes not only the state of the object but also QoS of the state. We discuss new equivalent and compatible relations among methods with respect to QoS. By using the relations, we newly discuss a QoS-based compensating way to recover the object from the less qualified state.

## 1. Introduction

Units of resources in distributed systems are referred to as *objects*[10]. An object is an encapsulation of data and methods for manipulating the data. CORBA[10] is getting a general framework to realize the interoperable applications. The system is required to be *flexible* in the change of the system environment and applications' requirements in addition to supporting the interoperability of autonomous objects.

The service supported by the object is characterized by parameters showing the *quality of service* (*QoS*) like frame rate and number of colors. Yoshida and Takizawa[13] model *movement* of a mobile object to be the change of QoS supported by the object. It is critical to discuss how to support QoS which satisfies the application's requirement in change of QoS supported by multimedia objects. In MPEG-4[8],[9] and MPEG-7, multimedia data is composed of multimedia objects each of which may support a different level of QoS.

An object supports applications with service through the methods. The method may change not only the state of the object but also QoS supported by the object. Relations among the methods are discussed so far with respect to the states of the objects. For example, a pair of methods are *equivalent* if the states obtained by applying the methods in any order are the same[1]. In this paper, we discuss kinds of relations among the methods with respect to QoS. Here, suppose that a state $s_2$ is obtained by dropping some frames in a state $s_1$ of a multimedia object. If $s_2$ satisfies the applications' requirements, $s_2$ is considered to be equivalent with $s_1$. In addition, there are two aspects of QoS, i.e., *state QoS* and *view QoS*. The state QoS means QoS which the state of the object intrinsically supports. The applications can view QoS of the object only through the methods. For example, suppose that a multimedia object supports higher quality image data and a *display* method. Here, the application can only view lower quality image if *display* can output only lower quality image. QoS viewed through *display* is *view* QoS of the object.

Effects done by methods computed have to be removed if applications' requirements are not satisfied, e.g., the system is faulty. The effects can be removed by the *compensation*[7],[12] of the methods computed. In multimedia applications, it takes time to restore a large volume of high-resolution video data. We can reduce time for recovering the system if data with lower resolution but satisfying the application requirement is restored instead of restoring the high-resolution data. In this paper, we discuss a compensation way where an object $o$ may not be rolled back to the previous state at which $o$ has been but can be surely rolled back to a state supporting QoS which satisfies the application's requirement. We can reduce time for rolling back the objects by this way.

In Section 2, we present a model of the system. In Sections 3 and 4, we discuss relations among the methods and the compensation on the basis of QoS, respectively.

## 2. System Model

### 2.1 Objects

A system is composed of multiple objects distributed on multiple computers which are interconnected by reliable communication networks. Each object is an encapsulation of data and a collection of abstract methods $op_1, \ldots, op_l$ only

† Department of Computers and Systems Engineering, Tokyo Denki University

by which $o_i$ can be manipulated. There are two kinds of objects, *class* and *instance.* A class gives a framework, i.e., set of attributes and collection of methods. An instance is created from the class, which is a tuple of values each of which is given to each attribute of the class. From here, let a term "object" mean an instance.

Methods change the state of an object $o$ and output data obtained from the state as the responses. Let $op_t(s)$ denote a state of the object $o$ obtained by applying a method $op_t$ to a state $s$ of $o$. A state means a tuple of values in an instance of $o$. $[op_t(s)]$ denotes the response obtained by applying $op_t$ to a state $s$ of $o$. For example, $[display(s)]$ shows image displayed on a monitor or printer from a state $s$ of a multimedia object by $display(s)$. $op_t \circ op_u$ means that a method $op_u$ is computed after another method $op_t$ is terminated. Here, a conflicting relation[7] among a pair of methods $op_t$ and $op_u$ is defined as follows: $op_t$ *conflicts* with $op_u$ if $op_t \circ op_u(s) \neq op_u \circ op_t(s)$, $[op_t(s)] \neq [op_u \circ op_t(s)]$, or $[op_t \circ op_u(s)] \neq [op_u(s)]$ for some state $s$ of $o_i$. For example, *record* conflicts with *delete* in the object *movie.* A method $op_t$ is *compatible* with $op_u$ unless $op_t$ conflicts with $op_u$ in the object $o$. The conflicting relation is not transitive. We assume the conflicting relation is symmetric. Let $\langle s \rangle$ denote a tuple $\langle [op_1(s)], \ldots, [op_l(s)] \rangle$ of the responses obtained from a state $s$, i.e., *view* of the state $s$ of an object $o$.

An object can be composed of other objects. For example, suppose one movie scene shows a person driving a car on a road. An object for the scene is composed of four objects showing a person, car, road, and background. In MPEG-4, a multimedia data is composed of multiple objects like audio/video objects (AVOs) and sound object.

### 2.2 Quality of Service (QoS)

Each object $o$ supports applications with some service. The service can be obtained by issuing methods supported by the object $o$. Each service is characterized by parameters like level of resolution, number of frames, and number of colors. Quality of service (QoS) supported by the object $o$ is given by the parameters. Even if a pair of objects support the same types of service, they may provide different levels of QoS.

The *scheme* of QoS is given in a tuple of attributes $\langle a_1, \ldots, a_m \rangle$ where each attribute $a_i$ shows a parameter. Let dom$(a_i)$ be a *domain* of an attribute $a_i$, i.e., a set of possible values to be taken by $a_i$ ($i = 1, \ldots, m$). For example, dom(*resolution*) is a set of numbers each of which shows the number of pixels for each frame. A QoS *instance* $q$ of the scheme $\langle a_1, \ldots a_m \rangle$ is given in a tuple of values $\langle v_1, \ldots, v_m \rangle \in \text{dom}(a_1) \times \ldots \times \text{dom}(a_m)$. Let $a_i(q)$ show a value $v_i$ of an attribute $a_i$ in $q$. The values in dom$(a_i)$ are partially ordered by a precedent relation $\preceq \subseteq \text{dom}(a_i) \times \text{dom}(a_i)$, i.e., a QoS value $v_1$ *precedes* another QoS value $v_2$ ($v_1 \succeq v_2$) in dom$(a_i)$ if $v_1$ shows better QoS than $v_2$. For example, $120 \times 100 \preceq 160 \times 120$ [pixels] for an attribute *resolution.* Let $q_1$ and $q_2$ show QoS instances of the scheme $\langle a_1, \ldots, a_m \rangle$. $q_1$ *totally dominates* $q_2$ ($q_1 \succeq q_2$) iff $a_i(q_1) \succeq a_i(q_2)$ for every attribute $a_i$. Let $A$ be a subset $\langle b_1, \ldots, b_k \rangle$ of the QoS scheme $\langle a_1, \ldots, a_m \rangle$ where each $b_k \in \{a_1, \ldots, a_m\}$ and $k \leq m$. A projection $[q]_A$ of the QoS instance $q$ on $A$ is $\langle w_1, \ldots, w_k \rangle$ where $w_i = b_i(q)$ for $i = 1, \ldots, k$. A QoS *instance* $q_1$ of a scheme $A_1$ *partially dominates* $q_2$ of $A_2$ iff $a(q_1) \succeq a(q_2)$ for every attribute $a$ in $A_1 \cap A_2$. $q_1$ *subsumes* $q_2$ ($q_1 \supseteq q_2$) iff $q_1$ partially dominates $q_2$ and $A_1 \supseteq A_2$. Let $S$ be a set of QoS *instances* whose schemes are not necessarily the same. A QoS instance $q_1$ is *minimal* in the set $S$ iff there is no QoS instance $q_2$ in $S$ such that $q_2 \preceq q_1$. $q_1$ is *minimum* in $S$ 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* in $S$ iff $q_2 \preceq q_1$ for every $q_2$ in $S$. $q_1 \cup q_2$ and $q_1 \cap q_2$ show a *least upper bound* and a *greatest lower bound* of QoS instances $q_1$ and $q_2$ in $S$ on $\preceq$, respectively. $q_1 \cup q_2$ is some QoS instance $q_3$ in $S$ such that 1) $q_1 \preceq q_3$ and $q_2 \preceq q_3$, and 2) there is no instance $q_4$ in $S$ where $q_1 \preceq q_4 \preceq q_3$ and $q_2 \preceq q_4 \preceq q_3$. $q_1 \cap q_2$ is defined similarly to $\cup$.

Applications require an object $o$ to support some QoS which is referred to as *requirement* QoS (*RoS*). Let $r$ be an RoS instance. Here, suppose an object $o$ supports a QoS instance $q = \langle v_1, \ldots, v_m \rangle$ where each $v_i$ is a value of the attribute $a_i$, i.e., $v_i \in \text{dom}(a_i)$. Here, let $A_r$ be the scheme of $r$ and $A_q$ be the scheme of $q$. The instance $q$ *subsumes* $r$ ($q \supseteq r$) iff $q$ partially dominates $r$ and $A_q \supseteq A_r$. If $q \supseteq r$, the applications can get enough service from $q$. Otherwise, $q$ is less qualified than $r$.

### 2.3 QoS of Object

QoS of an object $o$ has two asp cts: *state* QoS which is obtained from the state of $o$ and *view* QoS which is supported through the methods of $o$. For example, let us consider an ob-

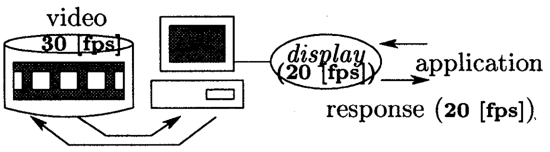


**Fig. 1** QoS of video object.

ject *video* with a *display* method as shown in **Fig. 1**. A state $s$ of the object *video* supports video data with a rate 30 [fps], which is a state QoS. $Q(s) = 30$ [fps]. However, *display* can display the view $[display(s)]$ on the monitor of the video data from the state $s$ only at a rate 20 fps. This is a view QoS. $Q([display(s)]) = 20$ [fps]. Here, there is a constraint "$Q([op_t(s)]) \preceq Q(s)$" for every method $op_t$ and every state $s$ of an object $o$. The object $o$ cannot support the applications with higher QoS than supported by the methods. If $Q([op_t(s)]) \prec Q(s)$ for some state $s$ of the object $o$, $op_t$ is *less qualified* in the object $o$. The method $op_t$ is *fully qualified* in $o$ if $Q([op_t(s)]) = Q(s)$ for every state $s$ of $o$. In Fig. 1, the method *display* is less qualified for the object *video*. Let maxQoS($op_t$) show the maximum QoS which $op_t$ can support, i.e. $Q([op_t(s)]) \preceq$ maxQoS($op_t$) for every state $s$ of the object $o$. Let $s_1$ and $s_2$ be states of an object $o$. The applications cannot differentiate states $s_1$ and $s_2$ if data viewed by applying a method $op_t$ to $s_1$ and $s_2$ are the same, i.e., $[op_t(s_1)] = [op_t(s_2)]$ in the object $o$.

**[Definition]** A state $s_1$ is *$op_t$-equivalent* with $s_2$ in an object $o$ iff $[op_t(s_1)] = [op_t(s_2)]$. □

$Q(\langle s\rangle)$ is defined to be a tuple $\langle Q([op_1(s)]), \ldots, Q([op_l(s)])\rangle$, i.e., view QoS of a state $s$ of an object $o$ which can be obtained through the methods. $Q(\langle s\rangle)$ shows QoS of $o$ which the applications can view through the methods.

**[Definition]** A state $s_1$ is *method-equivalent* with a state $s_2$ of an object $o$ iff $\langle s_1\rangle = \langle s_2\rangle$, i.e., $[op_t(s_1)] = [op_t(s_2)]$ for every method $op_t$ of $o$. □

Even if $s_1 \neq s_2$, the applications view a pair of states $s_1$ and $s_2$ of the object $o$ to be the same because the applications get the same response through every method. Let $maxQ_o$ denote maximum QoS to be supported by $o$, i.e., maximum of $Q(\langle s\rangle)$ for every state $s$ of $o$. Let $minQ_o$ denote minimum QoS of $o$.

A multimedia object *movie* supports the movie video including low-resolution image data (120 × 100 pixels) with a *display* method. A *hypermovie* object supports hyper video images of high-resolution (160 × 120 pixels) with more kinds of methods including *display*, *stop-motion*, *merge*, and *divide* than the object *movie*. A state $s_{movie}$ includes the low-resolution video image of a movie $m$. $s_{hypermovie}$ shows the high-resolution video image of multiple movies including $m$. Here, $Q(s_{hypermovie}) \succeq Q(s_{movie})$. *display* of *hypermovie* can display the high-resolution video image with multi-window while *display* of *movie* can just display the low-resolution video image. Here, $Q([display(s_{hypermovie})]) \succeq Q([display(s_{movie})])$. *hypermovie* supports higher quality of video image and more fruitful methods than *movie*.

Real objects in the real world have infinite level of QoS. In order to realize the real objects in computers, we have to reduce QoS of the objects. Thus, we model that each object state is realized by mapping the infinite level of QoS to the limited level of QoS depending on the facilities of the computers. The state of the real object is referred to as a *super state*. Let super($s$) denote a super state of a state $s$ of an object $o$ which is realized in the computer. Here, $Q(\text{super}(s)) \succeq Q(s)$. We assume that there exists exactly one super state for each state $s$. QoS of every super state is maximum.

**[Definition]** A state $s_1$ is *state-equivalent* with a state $s_2$ in an object $o$ iff super($s_1$) = super($s_2$). □

For example, suppose that a state $s_1$ of the object *video* supports video data of frame rate 30 [fps]. Suppose a new state $s_2$ is obtained by dropping some frames in the state $s_1$. If $s_2$ is state-equivalent with $s_1$, $s_1$ and $s_2$ are derived from a same super state by reducing QoS but they support different levels of QoS.

There are two aspects of objects to be considered, i.e., states and QoS of the objects. Hence, each object supports two types of primitive methods, one for manipulating the state of the object and the other one for manipulating QoS of the object. The former is a *state method* and the latter is a *QoS method*. The method *drop* is a QoS method because it only changes QoS of the object *video*. For a QoS method $op$, a state $op(s)$ is state-equivalent with every state $s$ of an object $o$, i.e., super($op(s)$) = super($s$). For a pair of QoS methods $op_t$ and $op_u$, $op_t(s)$ and $[op_t(s)]$ are state-equivalent with $op_u(s)$ and $[op_u(s)]$, respectively, for every state $s$ of an object $o$ because they only change the QoS of the object $o$. On the other hand,

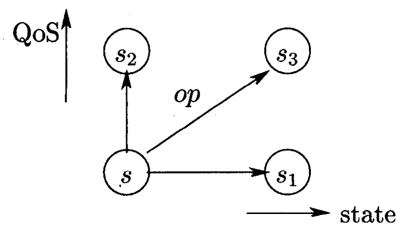


**Fig. 2** Transition diagram.

for a state method $op$, $Q(op(s)) = Q(s)$ while $s \neq op(s)$. Here, we introduce a transition diagram to show the change of states and QoS as shown in **Fig. 2**, where a node shows a state and a directed edge indicates a state transition. A horizontally directed edge $s \rightarrow s_1$ indicates that a state $s$ is changed to another state $s_1$ by a state method which manipulates the state of the object $o$. Here, QoS of the state $s_1$ is the same as the state $s$, i.e., $Q(s_1) = Q(s)$. On the other hand, a vertically directed edge $s \rightarrow s_2$ shows that a state $s_2$ is obtained from $s$ by changing QoS of $s$ through a QoS method. For example, $s_2$ is obtained by increasing number of colors of $s$. Applications can consider $s$ and $s_2$ to be the same except for the number of colors. That is, $s_2$ is state-equivalent with $s$. A public method is implemented by using these primitive methods, i.e., changes not only the state but also QoS of the state. In Fig. 2, an oblique edge $s \rightarrow s_3$ denotes that a method $op$ obtains a state $s_3$ by changing both state and QoS of the state $s$.

## 3. QoS Relation Among Methods

We discuss how methods $op_1, \ldots, op_l$ supported by an object $o$ are related with respect to QoS.

### 3.1 Equivalency

A method $op_t$ is *equivalent* with another method $op_u$ in an object $o$ iff $op_t(s) = op_u(s)$ and $[op_t(s)] = [op_u(s)]$ for every state $s$ of $o$. That is, the methods $op_t$ and $op_u$ not only output the same response data but also change the state of $o$ to the same state.

Suppose an object *movie* is composed of two subobjects, an *advertisement* object and a *content* object. The advertisement object is removed from the object *movie* by a method *delete*. An application does not care the difference between the original version and the updated version of *movie* since the application is interested only in the content part of *movie*. The updated version is *semantically equivalent*

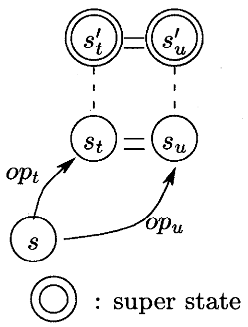


**Fig. 3** Semantically state-equivalent method.

with the original version because the two versions are considered to be the same from the application point of view. That is, the super states of the original and updated versions are considered to be the same. The two versions support the same QoS.

Suppose that a pair of super states $s'_t$ and $s'_u$ of an object $o$ are considered to be the same in some applications. Suppose $s_t = op_t(s)$ and $s_u = op_u(s)$ for a state $s$ of $o$. If $s'_t$ and $s'_u$ are super states of $s_t$ and $s_u$, respectively, i.e., $s'_t = \text{super}(s_t)$ and $s'_u = \text{super}(s_u)$. The states $s_t$ and $s_u$ are obtained by reducing QoS of $s'_t$ and $s'_u$. Here, $s_t$ and $s_u$ are *semantically equivalent* (**Fig. 3**). It is noted that $Q(s_t) = Q(s_u)$.

[**Definition**] A state $s_1$ is *semantically equivalent* with $s_2$ in an object $o$ iff super$(s_1)$ and super$(s_2)$ are considered to be the same by the application. □

[**Definition**] A method $op_t$ is *semantically equivalent* with another method $op_u$ in an object $o$ iff $op_t(s)$ is semantically equivalent with $op_u(s)$ and $Q(op_t(s)) = Q(op_u(s))$ for every state $s$ of $o$. □

Here, suppose the object *movie* supports two versions *old-display* and *new-display* of a method *display*. *new-display* can display the same video image as *old-display* while *new-display* can display at a faster rate than *old-display*. *new-display* is considered to be the same as *old-display* because they output the same image data and do not change the state of *movie*. However, they support different levels of QoS, i.e., *new-display* is more qualified than *old-display* with respect to the display speed. That is, $Q([\textit{old-display}(s)]) \preceq Q([\textit{new-display}(s)])$ for every state $s$ of *movie*.

[**Definition**] A method $op_t$ is *more qualified* than another method $op_u$ in an object $o$ iff $Q([op_t(s)]) \succeq Q([op_u(s)])$ and $op_t(s)$ is state-equivalent with $op_u(s)$ for every state $s$ of the

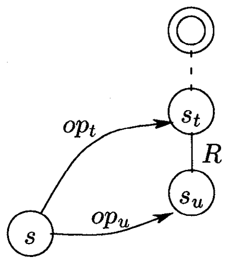


**Fig. 4** RoS-equivalent method.

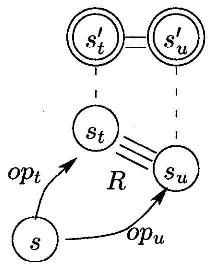


**Fig. 5** Semantically RoS-equivalent method.

object $o$. □

Let $R$ be QoS which an object is required to support for an application, i.e., RoS. The application does not mind which method *old-display* or *new-display* is used to display the movie if the application does not care the display speed in the object *movie*. Two methods *old-display* and *new-display* are considered to be equivalent with respect to $R$ if they support QoS subsuming $R$ even if $Q([\textit{old-display}(s_{movie})]) \neq Q([\textit{new-display}(s_{movie})])$ for a state $s_{movie}$ of the object *movie*.

**[Definition]** A state $s_t$ is RoS-equivalent with $s_u$ on RoS $R$ in an object $o$ ($s_t -_R s_u$) iff $Q(op_t(s)) \cap Q(op_u(s)) \supseteq R$ and $op_t(s)$ is state-equivalent with $op_u(s)$ for every state $s$ of $o$. □

**[Definition]** A method $op_t$ is *RoS-equivalent* with another method $op_u$ of an object $o$ on RoS $R$ iff $op_t(s)$ is RoS-equivalent with $op_u(s)$ for every state $s$ of $o$. □

In **Fig. 4**, $s_t = op_t(s)$ is state-equivalent with $s_u = op_u(s)$. If $Q(s_t)$ and $Q(s_u)$ satisfy $R$, $op_t$ and $op_u$ are RoS-equivalent. $op_t$ is more qualified than $op_u$ since $Q(s_t) \supseteq Q(s_u)$.

In the first example presented in this subsection, suppose that the updated version supports higher level of QoS than the original one. The versions are *semantically* and *RoS-equivalent.*

**[Definition]** A state $s_t$ is *semantically RoS-equivalent* with a state $s_u$ on RoS $R$ in an object $o$ ($s_t \equiv_R s_u$) iff super($op_t(s)$) is semantically equivalent with super($op_u(s)$) and $Q(op_t(s)) \cap Q(op_u(s)) \supseteq R$ for every state $s$ of $o$. □

**[Definition]** A method $op_t$ is *semantically RoS-equivalent* with a method $op_u$ of an object $o$ on RoS $R$ iff $op_t(s) \equiv_R op_u(s)$ for every state $s$ of $o$. □

In **Fig. 5**, $s_t = op_t(s)$ and $s_u = op_u(s)$, and $s'_t$ = super($s_t$) and $s'_u$ = super($s_u$). $s'_t$ is semantically equivalent with $s'_u$. $Q(s_t)$ and $Q(s_u)$ satisfy RoS $R$ while $Q(s_t)$ may not be the same as $Q(s_u)$. Here, $s_t$ is semantically RoS-equivalent with $s_u$ ($s_t \equiv_R s_u$).

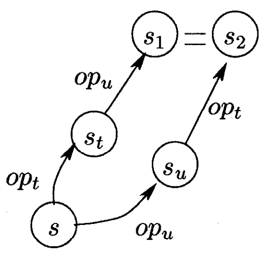


**Fig. 6** Semantically compatible method.

## 3.2 Compatibility

We discuss in which order a pair of methods $op_t$ and $op_u$ supported by an object $o$ can be computed in order to keep the object $o$ consistent. According to the traditional theory 1),7), a method $op_t$ *conflicts* with another method $op_u$ in an object $o$ iff the result obtained by computing $op_u$ after $op_t$ depends on the computation order. $op_t$ is *compatible* with $op_u$ unless $op_t$ conflicts with $op_u$.

**[Definition]** A method $op_t$ is *semantically compatible* with a method $op_u$ in an object $o$ iff $op_t \circ op_u(s)$ is semantically equivalent with $op_u \circ op_t(s)$ for every state $s$ of $o$. □

In **Fig. 6**, $s_1 = op_t \circ op_u(s)$ and $s_2 = op_u \circ op_t(s)$. Here, $s_1$ is semantically equivalent with $s_2$ since the super states of $s_1$ and $s_2$ are equivalent in the application. Hence, $op_t$ is semantically compatible with $op_u$. $Q(s_1) = Q(s_2)$. $op_t$ semantically conflicts with $op_u$ unless $op_t$ is semantically compatible with $op_u$.

Suppose a multimedia object $M$ displays MPEG-4 data. The MPEG-4 data has QoS of a frame rate 30 fps and 256 colors. A method *mediascaling* of $M$ reduces a frame rate to a half of the original one. On the other hand, a method *reduce* decreases a number of colors to 16 colors. The application can get the same QoS of a state obtained by applying *mediascaling* after *reduce* as in the reverse order. In any case, the application can get the MPEG-4 data with 15 fps and 16 colors.

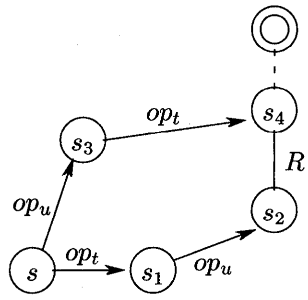


**Fig. 7** RoS-compatible method.

A multimedia data is composed of multiple objects in MPEG-4. Each object can be manipulated independently of the other component objects. Suppose a multimedia object $M$ displays MPEG-4 data which is composed of two objects showing colored background and car. A method *add* of the object $M$ takes an object *car* into the MPEG-4 data. On the other hand, a method *grayscale* changes a colored video object to a white-black gradation video. Suppose an application performs *grayscale* after *add*. The MPEG-4 data obtained by *add* and *grayscale* is a white-black gradation video with background and car. However, the MPEG-4 data obtained by applying *add* after *grayscale* is different from one obtained by applying *grayscale* after *add*. This MPEG-4 data includes white-black background and colored car objects. That is, QoS of a state of an object obtained by applying QoS methods depends on the application order of the methods.

**[Definition]** A method $op_t$ is *RoS-compatible* with $op_u$ on some RoS $R$ ($op_t \circ op_u(s) -_R op_u \circ op_t(s)$) in an object $o$ iff $op_t \circ op_u(s)$ is RoS-equivalent with $op_u \circ op_t(s)$ on $R$ for every state $s$ of $o$. □

In **Fig. 7**, $s_4$ is state-equivalent with $s_2$. That is, $s_2$ and $s_4$ have the same super state. $Q(s_2) \neq Q(s_4)$ but $Q(s_2)$ and $Q(s_4)$ satisfy $R$.

The RoS-compatibility relation is symmetric. Unless a method $op_t$ is RoS-compatible with another method $op_u$, $op_t$ *RoS-conflicts* with $op_u$. In the multimedia object $M$, the methods *reduce* and *mediascaling* are RoS-compatible. However, *add* RoS-conflicts with *grayscale*.

Suppose an application is not interested in how colorful movies are. A method *update* changes an object *movie* from a colored version to a monochromatic one. The colored *movie* $m$ is seen by performing *display*, i.e., [*display*($m$)]. If *update* is applied to the movie $m$, the monochromatic version of $m$ is seen. Since

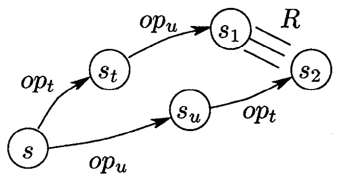


**Fig. 8** Semantically RoS-compatible method.

the application is not interested in the color of $m$, both versions are considered to satisfy the requirement QoS (*RoS*) required by the application. Hence, $Q([display(m)]) \cap Q([update \circ display(m)]) \supseteq R$ and $Q(display \circ update(m)) = Q(update \circ display(m))$. *display* and *update* are RoS-compatible. However, they are not semantically compatible because $Q([update \circ display(m)]) \neq Q([display(m)])$.

**[Definition]** A method $op_t$ is *semantically RoS-compatible* with $op_u$ in an object $o$ with respect to RoS $R$ iff $op_t \circ op_u(s)$ is semantically RoS-equivalent with ($\equiv_R$) $op_u \circ op_t(s)$ on $R$ for every state $s$ of $o$. □

In **Fig. 8**, $s_1 = op_t \circ op_u(s)$ and $s_2 = op_u \circ op_t(s)$ where $s_1$ and $s_2$ are semantically equivalent. $Q(s_1)$ and $Q(s_2)$ satisfy RoS $R$.

## 4. Compensation

A method $op_u$ is a *compensating* method of $op_t$ if $op_t \circ op_u(s) = s$ for every state $s$ of an object $o$[5),7)]. Let $s'$ be a state obtained by computing the method $op_t$ on a state $s$ of the object $o$, i.e., $s' = op_t(s)$. Here, $o$ can be rolled back to the state $s$ if the compensating method of *op* is computed on the state $s'$. For example, *append* is a compensating method of *delete*.

Let us consider the multimedia object *ME* with two movies $A$ and $B$ at state $s_1$, where it takes two hours to *play* each of $A$ and $B$ (**Fig. 9**). Suppose that $A$ and $B$ are *merged* into a movie $C$ at state $s_2$. Then, $C$ is *divided* into two movies $A'$ and $B'$ of state $s_3$. It takes one hour and half to *play* each of $A'$ and $B'$ at state $s_3$. Each of $A$ and $B$ is composed of advertisement and content parts of the movie. $A'$ and $B'$ include only the contents of $A$ and $B$, respectively. The advertisements of $A$ and $B$ are merged into $AB$. Here, $s_3$ is semantically equivalent with $s_1$. *divide* is a *semantically* compensating method of *merge*.

**[Definition]** A method $op_u$ is a *semantically compensating* method of $op_t$ iff $op_t \circ op_u(s)$ is semantically equivalent with every state $s$ of an object $o$ (**Fig. 10**). □

**[Definition]** A method $op_u$ is an *RoS-*

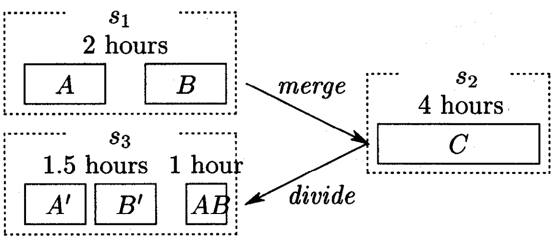


**Fig. 9** Example of semantically compensating method.

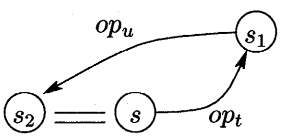


**Fig. 10** Semantically compensating method.

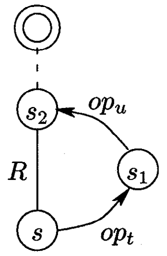


**Fig. 11** RoS-compensating method.

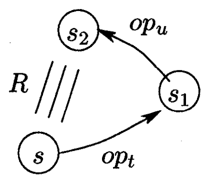


**Fig. 12** Semantically RoS-compensating method.

*compensating* method of a method $op_t$ in an object $o$ on RoS $R$ iff $op_t \circ op_u(s) \equiv_R s$ for every state $s$ of $o$ (**Fig. 11**). □

**[Definition]** A method $op_u$ is a *semantically RoS-compensating* method of $op_t$ in an object $o$ on RoS $R$ iff $op_t \circ op_u(s) \equiv_R s$ for every state $s$ of $o$ (**Fig. 12**). □

Suppose the multimedia object *ME* supports a method *divide2* which divides $C$ into three parts $A''$, $B''$, and $AB$ in addition to the methods *merge* and *delete* shown in **Fig. 13**. $A''$ and $B''$ are the content parts of $A$ and $B$, respectively, which are monochromatic at state $s_3$. $AB$ includes the advertisement parts of $A$ and $B$. $s_3$ denotes a state where $A''$, $B''$, and $AB$ are obtained from $A$ and $B$. $s_1$ and $s_3$ are not the same. Furthermore, $A$ and $B$ are col-

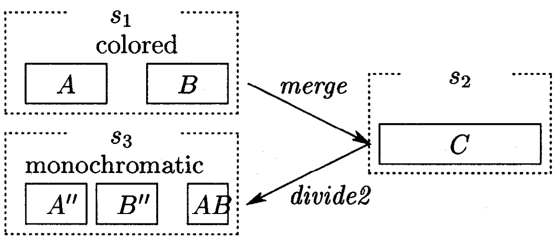


**Fig. 13** Example of semantically RoS-compensating method.

ored but $A''$ and $B''$ are monochromatic. That is, $Q(A) \supseteq Q(A'')$ and $Q(B) \supseteq Q(B'')$. Suppose an application just would like to see the monochromatic one. This is RoS $R$. Here, $Q(\langle s_3 \rangle) \supseteq R$. *divide2* is a *semantically RoS-compensating* method of *merge*.

## 5. Concluding Remarks

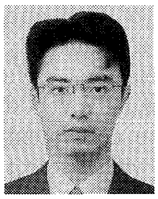
This paper has discussed how to make the distributed system flexible with respect to QoS supported by the objects. We have discussed the novel equivalent and conflicting relations among the methods on the basis of QoS and state, i.e., semantically, RoS, and semantically RoS equivalent and compatible relations. We have also discussed the compensating method to undo the work done. A state equivalent with the previous qualified state with respect to QoS is obtained by computing the compensating methods of methods computed.

## References

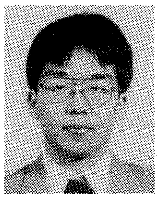
1) Bernstein, P.A., Hadzilacos, V. and Goodman, N.: *Concurrency Control and Recovery in Database Systems*, Addison-Wesley (1987).
2) Cambell, A., Coulson, G., Garcfa, F., Hutchison, D. and Leopold, H.: Integrated Quality of Service for Multimedia Communication, *IEEE InfoCom*, pp.732–793 (1993).
3) Campbell, A., Coulson, G. and Hutchison, D.: A Quality of Service Architecture, *ACM SIGCOMM Comp. Comm. Review*, Vol.24, pp.6–27 (1994).
4) Gall, D.: MPEG: A Video Compression Standard for Multimedia Applications, *Comm. ACM*, Vol.34, No.4, pp.46–58 (1991).
5) Garcia-Molina, H. and Salem, K.: Sagas, *Proc. ACM SIGMOD*, pp.249–259 (1987).
6) Kanezuka, T. and Takizawa, M.: QoS Oriented Flexibility in Distributed Objects, *Proc. Int'l Symp. on Communications (ISCOM'97)*, pp.144–148 (1997).
7) Korth, H.F., Levy, E. and Silberschalz, A.: A Formal Approach to Recovery by Compen-

sating Transactions, *Proc. VLDB*, pp.95–106 (1990).
8) MPEG Requirements Group: MPEG-4 Requirements, ISO/IEC JTC1/SC29/WG11 N2321 (1998).
9) MPEG Requirements Group: MPEG-4 Applications, ISO/IEC JTC1/SC29/WG11 N2322 (1998).
10) Object Management Group Inc.: The Common Object Request Broker: Architecture and Specification, Rev2.0 (1995).
11) Sabata, B., Chatterjee, S., Davis, M. and Syidir, J.J.: Taxonomy for QoS Specifications, *Proc. IEEE WORDS'97*, pp.100–107 (1997).
12) Takizawa, M. and Yasuzawa, S.: Uncompensatable Deadlock in Distributed Object-Oriented Systems, *Proc. IEEE ICPADS-92*, pp.150–157 (1992).
13) Yoshida, T. and Takizawa, M.: Model of Mobile Objects, *Proc. DEXA'96*, pp.623–632 (1996).

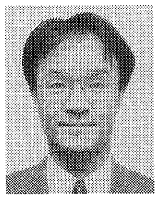
(Received May 8, 1998)
(Accepted September 7, 1998)



**Tetsuo Kanezuka** was born in 1974. He received his B.E. degree in computers and systems engineering from Tokyo Denki University, Japan in 1997. He is now a graduate student of the master course in the Dept. of Computers and Systems Engineering, Tokyo Denki Univ. His research interest is distributed multimedia networks.



**Hiroaki Higaki** was born in Tokyo, Japan, on April 6, 1967. He received the B.E. degree from the Dept. of Mathematical Engineering and Information Physics, the University of Tokyo in 1990. He is in the Dept. of Computers and Systems Engineering, Tokyo Denki Univ. He received the D.E. degree in 1997. His research interests include distributed algorithms and computer network protocols. He is a member of IEEE CS, ACM and IEICE.



**Makoto Takizawa** was born in 1950. He received his B.E. and M.E. degrees in Applied Physics from Tohoku University, 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. 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 was a program co-char of IEEE ICDCS-18, 1998 and serves on the program committees of many international conferences. His research interests include communication protocols, group communication, distributed database systems, transaction management, and security. He is a member of IEEE, ACM, IPSJ, and IEICE.