マルチメディア通信と分散処理　85－16
（１９９７．１１．６）

Quality-based Compensation in Flexible Distributed Systems

Tetsuo Kanezuka, and Makoto Takizawa
Tokyo Denki University
E-mail {kane, taki}@takilab.k.dendai.ac.jp

This paper discusses how to make a distributed system flexible so as to satisfy the application's requirement in the change of the system environment. The system is modeled to be composed of multiple objects which are cooperating. Each object supports other objects with types of service and quality of service (QoS). The change of the system is modeled to be the change of QoS supported by the objects. We discuss relation among the operations with respect to QoS. We define QoS-based equivalency and compatibility of the operations. By using the QoS-based relation, we newly discuss a QoS-based compensating way to recover from the less qualified state. Here, the object is transited to the state which can support QoS required but may not be the same as the previous one.

QoS に基づいた柔軟な分散型システム

金塚 哲郎　滝沢 誠
東京電機大学理工学部経営工学科
E-mail {kane, taki}@takilab.k.dendai.ac.jp

本論文では、システム環境の変更において、応用の要求を満足するために柔軟性のある分散型システムを構築する方法について論じる。システムはメッセージの交換により協調動作する複数のオブジェクトにより構成される。それぞれのオブジェクトは、それぞれのオブジェクトが持つサービスのタイプとクオリティオブサービス (QoS) をサポートする。システムの変化はオブジェクトによってサポートされる QoS の変化として見ることができる。あるオブジェクトは、障害や輻輳のようなオブジェクトの変化のため応用によって要求される QoS をサポートできないかもしれない。本論文では、より制限の少ない状態から復旧するための、QoS に基づいた補償方法について論じる。ここでは、オブジェクトは、以前の状態と同じではないかもしれないが、要求される QoS をサポートできる状態へ推移される。

1 Introduction

It is important to support applications with *flexible* service in a distributed system including kinds of commercially available workstations interconnected by standardized communication networks. In this paper, units of resources in the system are referred to as *objects*. Each object is modeled to be an encapsulation of data and operations for manipulating the data.

One of the major changes in the system is *fault*. There are two approaches to realizing the fault-tolerant system; replication and checkpointing. The active replication [2, 12] and passive replication [3] are discussed so far. In the replication methods, multiple replicas of an object cooperate to support services of the object. In the active replication, every replica does the same computation and communication. Hence, the service of the object can be supported as long as at least one replica is operational. In the checkpointing [6], the state of the object is saved in the stable log at the checkpoint. If the object is faulty, the object is rolled back to the checkpoint by restoring the state. Many protocols [6] to take the consistent checkpoints among the objects are discussed. The consistent checkpoint is defined to denote the states of the objects where no *orphan* messages exist. The *orphan* messages are ones received by an object but sent by no object. Tanaka and Takizawa [14] discuss an object-based checkpoint which allows orphan messages to exist but which is consistent from the object point of view. The number of checkpoints can be reduced. If some object is faulty, the object is rolled back to the object-based checkpoint.

In addition to the object fault, other properties of the system change. For example, the response time and throughput of an object o_i change up to the computation and communication load of o_i. The service supported by o_i is characterized by the parameters showing QoS. Yoshida and Takizawa [15] model the *movement* of the mobile object to be the change of QoS supported by the mobile object. It is critical to discuss how to support QoS which satisfies the application's requirement in the change of QoS supported by the objects. For example, if o_i does not support QoS required by the application, the application can use another object which supports enough QoS. o_i supports service through the operations which manipulate the state of o_i. In this paper, we discuss kind of relations among the operations supported by the object with respect to QoS. That is, QoS-based equivalency and compatibility of operations are defined. we discuss how the system supports the applications with QoS required in the change of QoS supported by the objects.

In this paper, we discuss a way to compute the *compensating* operations [9, 13] of the operations computed after c_i in order to roll o_i back to c_i. In addition, it is critical for o_i to support QoS required by the application when o_i is rolled back. In multimedia applications, it takes time to restore a large volume of high-resolution video data. Instead of restoring the high-resolution data, we can reduce the time for recovering the system if data with lower resolution but satisfying the application requirement is restored. In this paper, we discuss a method where o_i may not be rolled back to the same state denoted by c_i but can be surely rolled back to the state supporting QoS which satisfies the application's requirement.

In section 2, we present the model of the system. In section 3, we discuss the relation among the operations on the basis of QoS. In section 4, we discuss the compensation to recover the objects from the fault.

2 System Model

2.1 System configuration

A system is composed of multiple objects which are cooperating to achieve some objectives. Let O be a collection of objects, i.e. $O = \{o_1, \ldots, o_n\}$. The objects communicate with other objects by the reliable network.

Each object o_i is an encapsulation of the data structure δ_i and a collection τ_i of abstract operations $op_{i1}, \ldots, op_{il_i}$ for manipulating δ_i. o_i can be manipulated only through the operations supported by o_i. For example, the *bank* object supports *withdraw*, *deposit*, *check*, and *transfer* operations. The *movie* object supports *play*, *rewind*, *stop*, *quick-motion*, and *slow-motion* operations. The *movie* can be manipulated only by these operations.

Operations change the state of o_i and output some data as the responses. Let $op_{ij}(s_i)$ denote a state of o_i obtained by applying op_{ij} to a state s_i of o_i. $[op_{ij}(s_i)]$ denotes the response data obtained by $op_{ij}(s_i)$. $op_{ij} \circ op_{ik}$ means that op_{ik} is computed after op_{ij} Here, the conflicting relation [9] among the operations in τ_i is defined as follows: For every pair of operations op_{ij} and op_{ik} in τ_i, op_{ij} *conflicts* with op_{ik} if $op_{ij} \circ op_{ik}(s_i) \neq op_{ik} \circ op_{ij}(s_i)$, $[op_{ij}(s_i)] \neq [op_{ik} \circ op_{ij}(s_i)]$, or $[op_{ij} \circ op_{ik}(s_i)] \neq [op_{ik}(s_i)]$ for some state s_i of o_i. op_{ij} is *compatible* with op_{ik} unless op_{ij} conflicts with op_{ik}. If op_{ij} and op_{ik} are compatible, both the same state and the same response data are obtained independently of the computation order of op_{ij} and op_{ik}. We assume the conflicting relation is symmetric. For example, *read* and *write* conflict with one another in a *file* object. *deposit* and *withdraw* are compatible in the *bank* object.

2.2 Quality of service (QoS)

Each object o_i supports some service. The service can be obtained by issuing the operations supported by o_i. Each type of service is characterized by parameters like reliability, availability, security, cost, and performance. For example, the reliability is represented in terms of MTBF (mean time between failures). The performance is given in terms of response time and throughput. Quality of service (QoS) supported by o_i is given by the parameters. Even if two objects o_i and o_j support the same types of the service, they may provide different levels of QoS.

The *scheme* of QoS is given a tuple of attributes $\langle a_1, \ldots, a_m\rangle$ where each a_i shows a parameter. Let dom(a_i) be a set of possible values to be taken by a_i, named a *domain* of a_i. For example, for an attribute a_i showing MTBF, dom(a_i) is a collection of times. dom(a_i) is a set of resolutions in terms of pixels for a resolution attribution a_i. QoS *instance* of the scheme $\langle a_1, \ldots a_m\rangle$ is given in a tuple of the parameter values, i.e. $\langle v_1, \ldots, v_m\rangle \in \mathrm{dom}(a_1) \times \ldots \times \mathrm{dom}(a_m)$. The values in dom($a_i$) are partially ordered by a precedence relation $\preceq \subseteq \mathrm{dom}(a_i)^2$. For every pair of values v_1 and v_2 in dom(a_i), v_1 *precedes* v_2 ($v_1 \succeq v_2$) if v_1 shows better QoS than v_2. For example, 120 × 100 [pixels] $\preceq$ 160 × 120 [pixels] for the resolution. Let q_1 and q_2 show QoS $\langle v_{11}, \ldots, v_{1m}\rangle$ and $\langle v_{21}, \ldots, v_{2m}\rangle$ of the scheme $\langle a_1, \ldots, a_m\rangle$, respectively. q_1 is referred to as *totally dominate* q_2 ($q_1 \succeq q_2$) iff $v_{1i} \succeq v_{2i}$ for every $i = 1, \ldots, m$. Here, let $a_i(q)$ show a value v_i of a_i in $q = \langle v_1, \ldots, v_m\rangle$. Let A be a subset $\langle b_1, \ldots, b_k\rangle$ of $\langle a_1, \ldots, a_m\rangle$ where $b_k \in \{a_1, \ldots, a_m\}$ and $k \leq m$. A projection $[q]_A$ of q on A is $\langle w_1, \ldots, w_k\rangle$ where $w_i = b_i(q)$ for $i = 1, \ldots, k$. For every pair of QoS *instance* q_1 of a scheme A_1 and q_2 of A_2, q_1 *partially dominates* q_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 Q be a set of QoS *instance*. q_1 in Q is referred to as *minimal* in Q iff there is no q_2 in Q such that $q_2 \preceq q_1$. q_1 is *minimum* in Q iff $q_1 \preceq q_2$ for every q_2 in Q. q_1 is *maximal* iff there is no q_2 in Q such that $q_1 \preceq q_2$. q_1 is *maximum* in Q iff $q_2 \preceq q_1$ for every q_2 in Q. For every pair of q_1 and q_2 in Q, $q_1 \cup q_2$ and $q_1 \cap q_2$ show a *least upper bound* (*lub*) and a *greatest lower bound* (*glb*) of q_1 and q_2 on $\preceq$, respectively. $q_1 \cup q_2$ is some QoS q_3 in Q such that (1) $q_1 \preceq q_3$ and $q_2 \preceq q_3$, and (2) there is no q_4 in Q where $q_1 \preceq q_4 \preceq q_3$ and $q_2 \preceq q_4 \preceq q_3$. $q_1 \cap q_2$ is q_3 in Q such that (1) $q_3 \preceq q_1$ and $q_3 \preceq q_2$, and (2) there is no q_4 in Q where $q_3 \preceq q_4 \preceq q_1$ and $q_3 \preceq q_4 \preceq q_2$.

Users require an object to support some QoS which is referred to as *requirement* QoS (*RoS*). RoS is written in a tuple $\langle V_1, \ldots, V_k\rangle$ and V_i is a value of an attribute a_i. Here, suppose that an object o supports QoS $q = \langle v_1, \ldots, v_m\rangle$ where each v_i is a value of a_i. Here, let R be a tuple of QoS values $\langle V_1, \ldots, V_k\rangle$. A_R is the scheme of R, and A is the scheme of q. q *subsumes* R ($q \supseteq R$) iff q partially dominates R and $A \supseteq A_R$. If $q \supseteq R$, the users cat get enough service from o.

2.3 Multimedia objects

In this paper, we consider multimedia objects. QoS of o_i has two aspects, i.e. QoS obtained from the state s_i and QoS of the operations of o_i. They are named *state QoS* and *operation QoS*, respectively. For example, suppose that there are two objects o_i and o_j which have the same video data with high resolution and low resolution, respectively, which are compressed by MPEG [7]. Here, the state s_i of o_i has better QoS than the state s_j of o_j. o_i and o_j support a *play* operation which is realized by the decoder of the compressed state. If o_i and o_j support the low-level decoder, o_i and o_j support the same QoS by *play* even if o_i has the high-level resolution video data. Thus, QoS of o_i is given by the *state QoS* and the *operation*

QoS of o_i.

Each o_i supports a collection τ_i of operations $op_{i1}, \ldots, op_{il_i}$ for manipulating o_i. Let s_i denote a state of o_i. Let $Q(s_i)$ denote QoS of the state s_i of o_i. Let $Q(op_{ij})$ denote QoS supported by op_{ij} of o_i. QoS of o_i can be viewed through the operation of o_i. Here, let $Q([op_{ij}(s_i)])$ denote QoS viewed through op_{ij}, which is given to be minimum of $Q(s_i)$ and $Q(op_{ij})$. $Q\langle s_i\rangle$ is defined to be $\langle Q([op_{i1}(s_i)]), \ldots, Q([op_{il_i}(s_i)])\rangle$. Let $\langle s_i\rangle$ denote $\langle [op_{i1}(s_i)], \ldots, [op_{il_i}(s_i)]\rangle$, i.e. *view* of s_i. $Q(\langle s_i\rangle)$ shows QoS of o_i which the users can view through the operations. $Q(\langle s_i\rangle)$ *subsumes* $Q(\langle s_j\rangle)$ ($Q(\langle s_i\rangle) \supseteq Q(\langle s_j\rangle)$) iff there is some operation op_{ik} in o_i such that $Q([op_{ik}(s_i)]) \succeq Q([op_{ik}(s_j)])$ for every op_{ik} in o_j. Let s_i and s'_i be states of o_i. If $Q(s_i) \succeq Q(s_j)$, $Q(\langle s_i\rangle) \supseteq Q(\langle s_j\rangle)$. Let $maxQ(o_i)$ denote maximum QoS to be supported by o_i, i.e. maximum of $Q(s_i)$ for every state s_i of o_i. Let $minQ(o_i)$ denote minimum QoS of o_i. Here, $minQ(o_i) \preceq Q(s_i) \preceq maxQ(o_i)$ for every s_i of o_i.

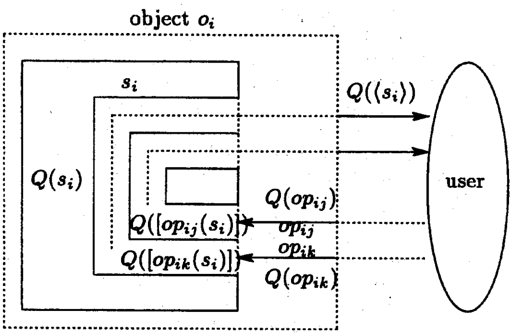


Figure 1: QoS of object.

[Definition] An object o_i *subsumes* o_j ($o_i \supseteq o_j$) iff $Q(\langle s_i\rangle) \supseteq Q(\langle s_j\rangle)$ for every pair of state s_i of o_i and s_j of o_j. □

Let us consider an example where there are two multimedia objects *movie* and *hypermovie*. The *movie* object supports the movie video including low-resolution image data (120 × 100 pixels) with a *display* operation. The *hypermedia* object supports hyper video images of high-resolution (160 × 120 pixels) with various kinds of operations including *display*, *stop-motion*, and *merge*. A state s_{movie} indicates the low-resolution video image of some movie m. A state $s_{hypermovie}$ shows the high-resolution video image of multiple movies including m. Here, $Q(s_{hypermovie}) \succeq Q(s_{movie})$. *hypermedia* supports more kinds of operations than *movie*. *display* of *hypermedia* can display the high-resolution video image with multi-window while *display* of *movie* can just display one low-resolution video image on the monitor. Here, $Q([display(s_{hypermedia})]) \succeq Q([display(s_{movie})])$. *hypermovie* $\supseteq$ *movie* since the *hypermovie* object supports higher quality of video image and more fruitful operations than *movie*.

Some operations op_{ij} change the state of o_i. Suppose that op_{ij} inserts some data d_{ij} to the state s_i of o_i. If $Q(s_i) \preceq Q(d_{ij})$, d_{ij} can be added to s_i. We consider case that $Q(s_i) \succ Q(d_{ij})$ [Figure 2(1)].

Since QoS of d_{ij} is worse than s_i, d_{ij} cannot be inserted in s_i. However, users can get service from o_i through operations of o_i. If QoS of d_{ij} viewed by an operation op_{ij} subsumes $Q(s_i)$, the user have no problem even if d_{ij} is inserted in s_i [Figure 2(2)].

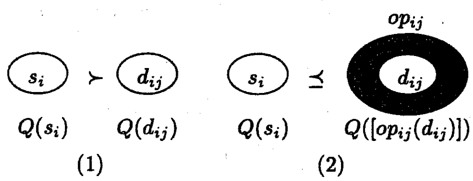


Figure 2: QoS viewed through op_{ij}.

In this paper, we consider a multimedia object *ME* where the multimedia data is edited as shown in Figure 3. *ME* is composed of subfunctions, *play* and *edit* functions. The *play* function supports operations which do not change the state of *ME*. The *edit* function supports operations which change the state. The *play* function supports types of operations; *multi-story*, *multi-aspect*, *multi-language*, *multi-voice*, and *multi-angle*. The *edit* function supports types of operations; *divide*, *combine*, *erase*, *all-erase*, and *move*.

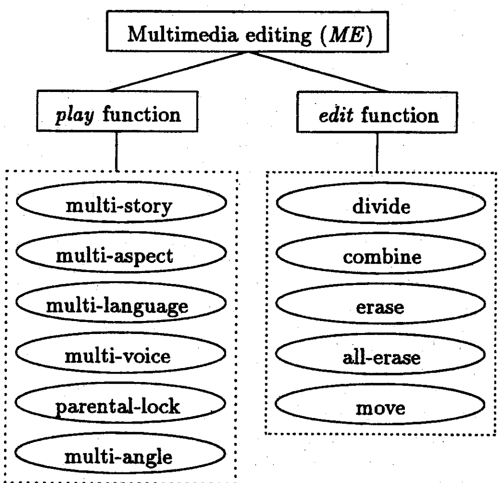


Figure 3: Multimedia editing (*ME*) system.

Each service is characterized by the following QoS parameters:

(1) CPU speed: MIPS.
(2) resolution: number of pixels, e.g. 160 × 128 pixels.
(3) number of frames per second: fps.
(4) color: number of colors for each pixel, e.g. 256 colors.

(5) sound: sampling frequency, e.g. 44.1kHz.
(6) reliability and availability: MTBF and MTTR [msec].
(7) security policy: mandatory or discretionary access control.
(8) accountability: identification and authentication.
(9) assurance: operational assurance and life-cycle assurance.

Let us consider Figure 4 where the *play function* is applied to *ME*. QoS of the *play* is characterized by CPU speed, color, resolution, and fps.

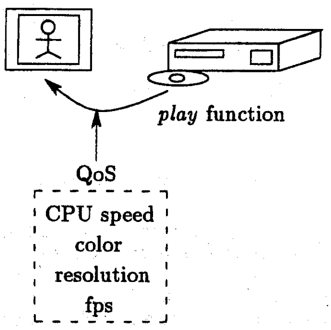


Figure 4: Multimedia editing (*ME*) system.

3 QoS Relations among Operatins

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

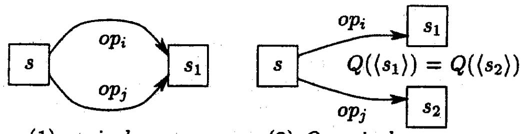
3.1 Equivalency

First, we discuss equivalent operations supported by o. An operation op_i is referred to as *equivalent* with op_j iff $op_i(s) = op_j(s)$ and $[op_i(s)] = [op_j(s)]$ for every state s of o [Figure 5(1)]. That is, op_i and op_j not only output the same data but also change the state of o to the same state. For example, suppose that there are two versions *old-display* and *new-display* of the *display* operation supported by the *movie* object. The new version *new-display* can display the same video image as the *old-display* operation while *new-display* can display faster than *old-display*. Here, *new-display* is equivalent with *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 better than *old-display*. We define a novel equivalent relation among the operations with respect to QoS supported by the operations.

[Definition] An operation op_i is *QoS-equivalent* (*Q-equivalent*) with op_j iff $Q(\langle op_i(s)\rangle) = Q(\langle op_j(s)\rangle)$ for every state s of an object o. □

That is, $op \circ op_i(s)$ and $op \circ op_j(s)$ supports the same view for every operation op [Figure 5(2)]. op_i is *Q-equivalent* with op_j if $Q(\langle op_i(s)\rangle) = Q(\langle op_j(s)\rangle)$.

Let R be RoS which an application requires an object o to support. The application does not mind which operation *old-display* or *new-display*



(1) *equivalency* (2) *Q-equivalency*

Figure 5: Equivalent operations.

is used if the user would like to see the movie independently of the display speed. Two operations are considered to be equivalent if they support QoS subsuming R even if $Q(old\text{-}display(s_{movie})) \neq Q(new\text{-}display(s_{movie}))$.

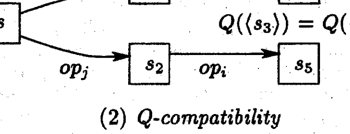
[Definition] An operation op_i is *RoS-equivalent* (*R-equivalent*) with op_j on R iff $Q(\langle op_i(s)\rangle) \cap Q(\langle op_j(s)\rangle) \supseteq R$. □

3.2 Compatibility

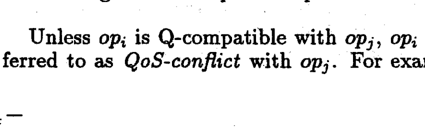
Next, we discuss in which order two operations op_i and op_j supported by the object o can be computed in order to keep the state of o consistent. According to the traditional theory [1,9], op_i and op_j conflict if the result obtained by computing op_j after op_i is different from op_i after op_j. op_i is *compatible* with op_j unless op_i conflicts with op_j [Figure 6(1)]. For example, suppose a *movie* object m is composed of an advertisement and a body. m is manipulated by the *delete* operation which removes the advertisement part from m. Even if the user sees the movie m after the advertisement part of m is removed by *delete*, the user does not care the difference between the original version of m and the updated version if the user is interested only in the content body part of m. That is, the updated version of m supports the same level of QoS as the original version of m.

We now define a QoS-compatible relation among the operations op_i and op_j.

[Definition] An operation op_i is *QoS-compatible* (*Q-compatible*) with op_j iff $Q(\langle op_i \circ op_j(s)\rangle) = Q(\langle op_j \circ op_i(s)\rangle)$ for every state s of an object o.□



(1) *compatibility*



(2) *Q-compatibility*

Figure 6: Compatible operations.

Unless op_i is Q-compatible with op_j, op_i is referred to as *QoS-conflict* with op_j. For example, suppose that an operation *delete* removes some frames from the *movie*. The movie can be seen only by the display operation with the low-level decoder. Here, the users can see the movie with the same quality even after *delete* is applied to the movie. Here, *delete* and *display* are Q-compatible.

The compatibility relation among the operations op_i and op_j depend on the requirement QoS (*RoS*) R required by the user.

[Definition] An operation op_i is *RoS-compatible* (*R-compatible*) with op_j on R iff $Q(\langle op_i(s)\rangle) \cap Q(\langle op_j \circ op_i(s)\rangle) \supseteq R$, $Q(\langle op_i \circ op_j(s)\rangle) \cap Q(\langle op_j(s)\rangle) \supseteq R$, and $Q(\langle op_i \circ op_j(s)\rangle) \cap Q(\langle op_j \circ op_i(s)\rangle) \supseteq R$. □

Suppose that a user is not interested in how colorful the movies are. Let *update* be an operation to change a movie from a colored version to a monochromatic one. Suppose that the *movie* object supports a colored movie m. The user sees the colored movie m by the operation *display*, i.e. $[display(m)]$. If m is changed by the *update* operation, the user sees the monochromatic version of m. Since the user is not interested in the color of m, both versions are considered to satisfy the requirement QoS (*RoS*) required by the user. 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 R-compatible. However, they are not Q-compatible because $Q([update \circ display(m)]) \neq Q([display(m)])$.

4 Compensation

Each object o_i may be faulty. We discuss how the object o_i recovers from the fault. In the traditional system, o_i is rolled back to the previous consistent state saved in the log l_i at the checkpoint c_i if o_i is faulty. If o_i is faulty, o_i is rolled back to c_i where the state stored in l_i is restored in o_i and then o_i is restarted. Many protocols [6] for taking the consistent checkpoints and restarting the processes are discussed so far.

Another way is to compute some operations to remove the effect done by the operations computed. op_j is a *compensating* operation of op_i if $op_i \circ op_j(s) = s$ for every state s of o [8, 9]. Let $\tilde{op}_i$ denote a compensating operation of op_i. Let s' be a state obtained by computing op_i on a state s of o. Here, o can be rolled back to s if $\tilde{op}_i$ is computed on s'. For example, *append* is a compensating operation of *delete*. *withdraw* is a compensating operation of *deposit*. Suppose that o computes a sequence of operations $op_1, \ldots, op_m$. In order to undo the operations $op_1, \ldots, op_m$, a sequence of the compensating operations $\tilde{op}_m, \ldots, \tilde{op}_1$ can be computed. That is, $op \circ op_1 \circ \ldots \circ op_m \circ \tilde{op}_m \circ \ldots \circ \tilde{op}_1 = op$ for every operation op. Here, $\tilde{op}_m \circ \ldots \circ \tilde{op}_1$ is a compensating sequence of $op_1 \circ \ldots \circ op_m$.

A pair of states s and s' of o may be considered to be the same from the application point of view even if $s \neq s'$. For example, suppose there are two accounts A and B. First, $A = 100$ and $B = 50$ at a state s_1. Suppose that $A = 110$ and $B = 40$ at s_2 after A and B are manipulated. If the application is only interested in the total amount of A and B, s_2 is considered to be equivalent with s_1. Thus, two states s and s' of o are *equivalent* ($s \equiv s'$) iff the application considers that s' is the same as s. op_j is a *semantically* compensating operation of op_i if $op_i \circ op_j(s_i) \equiv s$ for every state s of o [13]. Here, it is noted that $op_i \circ op_j(s)$ may not be s. $\hat{op}_i$ denotes a semantically compensating operation of op_i. For example, an operation t_1 transfers money a_1 from an account A to B. t_2 transfers money b_1 from B to A. For every state s, $t_1 \circ t_2(s) = s$ if $a_1 = b_1$. Here, t_2 is a compensating operation $\tilde{t}_1$. As presented here, suppose that the application is only interested in the amount of A and B. The amount of money in A and B is not changed after t_1 and t_2 are applied in any order, but the states obtained are different if $a_1 \neq b_1$. Here, t_2 is a semantically compensating operation of t_1, i.e. $t_1 \circ t_2(s) \equiv s$.

Here, suppose that a state s_2 is obtained by applying an operation op_{ij} to a state s_1 of an object o. Each state s_1 of o supports QoS $Q(s_1)$. Let us consider how to roll the object o back to the previous state s_1 from s_2. One way is to compute the compensating operation $\tilde{op}_i$ of op_i on s_1 since $op_i \circ \tilde{op}_i\,(s_1) = s_1$. Here, suppose that there exists an operation op_j such that $op_i \circ op_j(s_1) = s_3$ where $s_1 \neq s_3$ but $Q(s_3) \supseteq Q(s_1)$. s_3 is not the same as s_1 but supports better QoS than s_1. Here, s_2 is referred to as *QoS-equivalent* with s_1.

[Definition] An operation op_j is referred to as *QoS-compensating* (*Q-compensating*) operation of op_i iff $Q(\langle op_i \circ op_j(s)\rangle) = Q(\langle s\rangle)$ for every state s of an object o. □

Let $\hat{op}_i$ denote the *QoS-compensating* operation of op_i [Figure 7]. op_j is a kind of the semantically compensating operation of op_i, i.e. $\tilde{op}_i$ is $\hat{op}_i$.

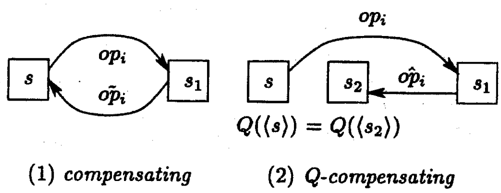


Figure 7: Compensating operation.

Let us consider the multimedia object *ME* as shown in Figure 8. Here, suppose that there are two movies A and B where it takes two hours to *play* each of A and B. This state is s_1. Suppose that a movie C is obtained by combining A and B through the *combine* operation. Here, the state is referred to as s_2. Then, C is divided into two movies A' and B' by the *divide* operation. The length of A' is one hour and half while B' is two hours and half. The state is named s_3. A is composed of some advertisement and the contents of the movie. A' includes only the contents of A. The advertisement of A is attached in B'. B' is also considered to be the same as B. Here, s_3 is QoS-equivalent with s_1 since $Q(\langle s_3\rangle) = Q(\langle s_1\rangle)$. *divide* is a QoS-compensating operation of *combine*.

In Figure 8, suppose one movie C is obtained by *combining* the movies A and B. Suppose the multimedia object *ME* supports an operation *divide2* which divides C into three parts A'', B'',

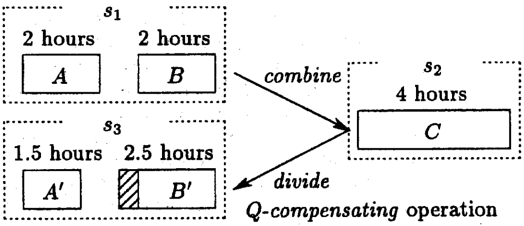


Figure 8: *Q-compensating* operation.

and AB. A'' and B'' are the content body parts of A and B, respectively, which are monochromatic. AB includes the advertisement parts of A and B. A state including A'', B'', and AB is named s_4. s_1 and s_4 are not the same. Furthermore, A and B support the colored movie but A'' and B'' support only the monochromatic one. That is, $A \supseteq A''$ and $B \supseteq B''$. Here, suppose that a user has a requirement QoS (RoS) R that it is all right for the user to see the monochromatic one. Here, $Q(\langle s_4 \rangle) \supseteq R$ [Fig. 9].

[Definition] An operation op_i is a *RoS-compensating* (*R-compensating*) operation of op_j on R iff $Q(\langle op_i \circ op_j(s) \rangle) \cap Q(\langle s \rangle) \supseteq R$ for every state s of an object o. □

divide2 is an example of the *R-compensating* operation of *combine*.

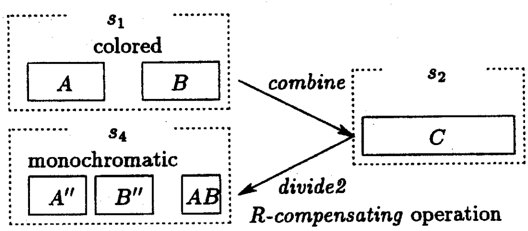


Figure 9: *R-compensating* operation.

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 operations on the basis of QoS. We have also discussed the compensating method to recover from the fault of the object. The object recovers from the fault by transiting to the state equivalent with the previous consistent state with respect to QoS by the compensating operations.

References

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

[2] Birman, K., P, Kenneth., and Renesse, V. Robbert., "Reliable Distributed Computing with the Isis Toolkit," *IEEE Comp. Society Press*, 1994.

[3] Budhiraja, N., Marzullo, K., Schneider, B. F., and Toueg, S., "The Primary-Backup Approach," *Distributed Computing Systems, ACM Press*, 1994, pp.199–221.

[4] Cambell, A., Coulson, G., Garcfa, F., Hutchison, D., and Leopold, H., "Integrated Quality of Service for Multimedia Communication," *IEEE InfoCom*, 1993, pp.732–793.

[5] Campbell, A., Coulson, G., and Hutchison, D., "A Quality of Service Architecture," *ACM SIGCOMM Computer Communication Review*, Vol. 24, 1994, pp.6–27.

[6] Chandy, K. M., Misra, J., and Haas, L. M., "Distributed Deadlock Detection," *ACM TODS*, Vol.1, No.2, 1983, pp.144-156.

[7] Gall, D., "MPEG: A Video Compression Standard for Multimedia Applications," *Comm. ACM*, Vol.34, No.4, 1991, pp.46–58.

[8] Garcia-Molina, H. and Salem, K., "Sagas," *Proc. of the ACM SIGMOD*, 1987, PP.249–259.

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

[10] Sabata, B., Chatterjee, S., Davis, M., and Syidir, J. J., "Taxonomy for QoS Specifications," *Proc of the IEEE Computer Society 3rd International Workshop on Object-oriented Real-time Dependable Systems* (*WORDS* '97), 1997, pp.1–10.

[11] Schmidt, D. C., Gokhale, A. S., Harrison, T. H., and Parulkar, G., "A High-Performance End System Architecture for Real-Time CORBA," *IEEE Communications Magazine*, 1997, pp.72–77.

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

[13] Takizawa, M., and Yasuzawa, S., "Uncompensatable Deadlock in Distributed Object-Oriented Systems," *Proc. of Int'l Conf. on Parallel and Distributed Systems* (*ICPADS'92*), 1992, pp.150–157.

[14] Tanaka, K., and Takizawa, M., "Distributed Checkpointing Based on Influential Messages," *Proc. of the 4th IEEE Int'l Conf. on Parallel and Distributed Systems* (*ICPADS-96*), 1996, pp. 440–447.

[15] Yoshida, T., and Takizawa, M., "Model of Mobile Objects," *Proc. of the 7th Int'l Conf. on Database and Expert Systems Applications* (*DEXA'96*), 1996, pp. 623–632.