マルチメディア通信と分散処理　66－15
（１９９４．　７．　８）

グループ通信における情報流制御

三田 浩也、 滝沢 誠
東京電機大学理工学部

グループウエア等の分散型応用では、複数エンティティ間でのグループ通信が要求される。グループ通信では、データ単位の順序性及び原子性に加えて、安全なグループ通信が要求される。また、各エンティティは、全エンティティではなく、グループ内の一部のエンティティにだけデータ単位を送信するかもしれない。そのような選択的なグループ通信では、もし送信されたデータ単位が宛先でないエンティティに対して送信されるならば、情報流は不正である。本論文では、束モデルを基本とした情報流モデルを利用して、グループ内での情報流を取り扱うための方式を論じる。

Information Flow Control in Group Communications

Hiroya Mita and Makoto Takizawa
Tokyo Denki University
Ishizaka, Hatoyama, Saitama 350−0, Japan
e−mail {mita, taki}@takilab.k.dendai.ac.jp

Distributed applications like groupware require group communications among multiple communication entities in a group. In addition to supporting the atomic and ordered delivery of messages to the entities in the group, secure group communication has to be supported. In some applications, each entity would like to send messages to any subset of the group, not necessarily all the entities, i.e. *selective* group communication. If a message received by an entity is forwarded to entities which are not the destinations, information in the message is illegally flown into the non-destination entities. In this paper, we would like to discuss how to deal with the information flow in the group by using the lattice-based information flow model.

1 Introduction

Distributed applications like groupware [5] are realized by the cooperation of multiple entities. An *entity* means a process or computer. Here, group communications among multiple communication entities in a group is required rather than the conventional one-to-one communications supported by *TCP/IP* [4] and OSI protocols [6]. In the group communications [2, 7, 8, 11, 12], every message sent by each entity has to be delivered to either all the entities or none of them in the group, i.e. *atomic delivery* of messages. Each entity also has to receive messages in some order [7, 8, 10]. For example, each entity receives the same messages in the same order, i.e. *total order* [8], and receives messages in the *causal order* [2, 9].

In addition to supporting the atomic and ordered delivery of messages, it has to be guaranteed that each entity receives messages from only the entities in the group, i.e. *authenticity*, and only the entities in the group send the massages in the group, i.e. *secrecy*. [13] discusses how only and all the proper entities in the group make an agreement on a common secret key by exchanging information ciphered by the public key system in order to support the authenticity and secrecy in the group communication.

In distributed applications, each entity would like to send messages to any subset in the group at any time, not necessarily to all the entities in the group. It is referred to as the *selective group* communication [7]. In the selective group communication, after receiving messages, the entities are not allowed to forward them to the non-destination entities. After receiving message p from entity E_i, if E_j forwards p to another E_k in the group, E_k can receive p from E_i. Even if E_i sends p to E_j but not E_k, E_k receives p from E_j. It is illegal information flow from E_j to E_k. When considering the secure group communication, the illegal information flow has to be prevented in addition to realizing the secrecy and authenticity in the group communication. The *lattice-based information flow model* [3, 10] is discussed to be a model for keeping all the information flows legal. Each E_i is assinged with security class. E_i can send messages to E_j if the class of E_i *precedes* the class of E_j. The set of security classes partially ordered by the precedence relation constructs a lattice. [1] presents the *mandatory* model where the relation between access control and information flow is discussed. If E_i belongs to some group, E_i is allowed to issue some kinds of primitives, e.g. *send* and *receive*, in the group. For example, suppose that E_i may receive messages but not send messages in the group. We would like to discuss how the primitives in the group are related with the information flow.

Each E_i may join multiple groups G_1, ..., G_n. E_i may play different roles in different groups. Through E_i, sensitive information in some G_j may be flown into another G_k if E_i forwards it from G_j to G_k. On the other hand, E_i in some group may send message to another group to which E_i does not belong. Thus, information may be flown from one group to another. In this paper, we would like to discuss such *inter-cluster* information flow and give rules to keep the inter-cluster information flow legal.

In section 2, we present a model of the communication system. In section 3, we present the lattice model of security classes. In section 4, we discuss the data transmission procedure on the basis of the security classes. In section 5, we discuss how to control the information flow among multiple groups.

2 System Model

The communication system is composed of *application*, *system*, and *network* layers [Figure 1]. The network layer provides the system entities with the *reliable* high-speed broadcast communication [7, 8, 9, 11, 12]. The entities at the system layer can communicate with each other by using the network layer to provide the application entities with the secure group communication. Application entity A_i takes the service through system service access point (SAP) S_i supported by system entity E_i. A *cluster* C is a set of the system SAPs S_1, ..., S_n, which is an extension of the conventional one-to-one *connection* concept [6] among two SAPs. C is referred to as *supported* by E_1, ..., E_n, written as $\langle E_1, ..., E_n \rangle$. E_i is referred to as *support* C. In this paper, we discuss how to support a group of the application entities with the secure information flow by using the reliable broadcast service of the network layer.

In the group communication [2, 8, 12], each message p sent by entity E_i is delivered to all the entities in C. [7] discusses a selective group communication where E_i can send each message p to any subset of C, not necessarily all the entities in C. Here, $p.DST$ denotes a set of destinations of p, and $p.DATA$ shows information carried by p. In this paper, we assume that the network layer supports the selective secure group communication. The application entities A_1, ..., A_n first reguest the system layer to establish a cluster C among them. C is established by the cooperation of the system entities E_1, ..., E_n. Then, each A_i selectively broadcasts messages to the destinations in C by using the selective secure group communication supported by the network layer. That is, A_i can send each message to only and all the destinations in C, i.e. *secrecy*, and can receive messages destined to A_i from only the entities in C, i.e. *authenticity*. [13] discusses how to realize the secure group communication [13] by using the public key system.

If A_i forwards message p to another A_j after receiving p, information carried by p is illegally flown into A_j if A_j is not the destination of p. In addition to realizing the secrecy and authenticity of the group communication, the informa-

tion flow among the application entities has to be controlled, i.e. messages not be forwarded to the entities which are not the destinations. In this paper, we would like to discuss how to provide the application entities with the secure information flow by using the reliable broadcast network supported by the network layer.

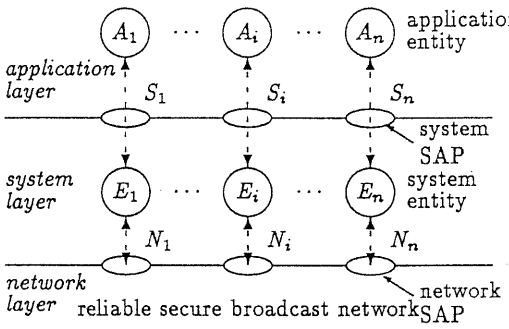

Figure 1: System model

3 Lattice-Based Model

We would like to present briefly a lattice-based model [3, 10] to deal with the information flow. Let S be a set of security classes. Every entity belongs to one security class. Information in each entity has the security class of the entity. The *can-flow* relation $\rightarrow$ is defined as a partially ordered relation on S, i.e. $\rightarrow \subseteq S^2$. For every pair of security classes s_1 and s_2 in S, information of s_1 can be flown into entities of s_2 iff $s_1 \rightarrow s_2$. That is, information in one entity of security class s_1 can be stored in another entity of s_2 iff $s_1 \rightarrow s_2$. For example, suppose that an individual p has a security class s_p and a database D has a security class s_D. If $s_D \rightarrow s_p$, p can obtain the data in D. The information flow model [3] is represented in a lattice $\langle S, \rightarrow, \cup, \cap \rangle$ where $\cup$ is a least upper bound (*lub*) and $\cap$ is a greatest lower bound (*glb*) on $\rightarrow$. For every pair of security classes s_1 and s_2 in S, *lub* of s_1 and s_2, i.e. $s_1 \cup s_2$ is defined to be s in S such that $s_1 \rightarrow s$, $s_2 \rightarrow s$, and there is no s_3 in S such that $s_1 \rightarrow s_3$, $s_2 \rightarrow s_3$, and $s_3 \rightarrow s$. $s_1 \cap s_2$ is defined in the same way. Here, $s_1 \succ s_2$ if $s_2 \rightarrow s_1$ but not $s_1 \rightarrow s_2$. s_1 is referred to as *dominate* s_2 ($s_1 \succeq s_2$) iff $s_1 \succ s_2$ or $s_1 = s_2$. $s_1 \succeq s_2$ means that information of s_1 is more sensitive than s_2. s_1 and s_2 are *comparable* iff $s_1 \preceq s_2$ or $s_2 \preceq s_1$.

Suppose that a cluster C supports application entities A_1, ..., A_n. Each A_i is supported by system entity E_i. Each A_i has one security class $class(A_i) \in S$. Each message p sent by A_i has a security class $class(p)$ which is the same as $class(A_i)$. A_i can send message to A_j if $class(A_i) \preceq class(A_j)$. Since $class(p) \preceq class(A_j)$, $p.DATA$ can be stored in A_j.

[Example 1] Suppose that there are three application entities A_1, A_2, and A_3 supported by a cluster C, whose security classes are s_1, s_2, and s_1, respectively. Suppose that there is a *can-flow* relation $s_1 \rightarrow s_2$ [Figure 2]. A_1 and A_3 can send messages to A_2, but A_2 can send messages to neither A_1 nor A_3 because $s_1 \preceq s_2$. A_1 and A_3 can communicate with each other because $class(A_1) = class(A_3)$. □

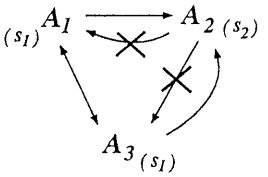

Figure 2: Information flow

[Definition] Let C be a cluster supporting application entities A_1, ..., A_n. The information flow in C is *legal* iff for every pair of A_i and A_j and for every message p sent by A_i to A_j in C, $class(A_i) \preceq class(A_j)$. □

Unless $class(A_i) \rightarrow class(A_j)$, the information flow in C is illegal if messages are delivered to A_j from A_i. In the secure group communication, every information flow among the application entities in C has to be kept legal.

[1] discusses the *mandatory access control* based on the information flow lattice, where entity E_i can read information in E_2 if $class(E_1) \succeq class(E_2)$, and E_1 can write information in E_2 if $class(E_1) \succeq class(E_2)$. The former is the simple access property, and the latter is the *-property.

4 Roled Cluster

4.1 Roles

We would like to redefine a cluster C to be a tuple of roles $\langle R_1, ..., R_n \rangle$ to deal with the information flow. Let S be a set of security classes. Let O be a set of primitives which application entities can issue to C, e.g. *send* and *receive* primitives. Each role R_i is defined to be a pair of a security class $s_i (\in S)$ and a collection $O_i (\subseteq O)$ of primitives which application entities can issue to C, i.e. $R_i = \langle s_i, O_i \rangle$. Let $class(R_i)$ denote s_i and $Op(R_i)$ denote O_i. Suppose that application entities A_1, ..., A_n establish C where each A_i is supported by system entity E_i. Each A_i is referred to as *bound* to C with R_i if C is established by the cooperation of E_1, ..., E_n. It is written as $\langle A_1{:}R_1, ..., A_n{:}R_n \rangle$ named an *instance* of C which denotes a state of C being established. This means that each A_i plays a role R_i in C, i.e. A_i can issue primitives in $Op(R_i)$ to C. There are the following primitives for C, i.e. *send*, *receive*, *open*, *close*, *abort*, and *reset* primitives. The *open* primitive is issued to

establish a cluster. On receipt of the open primitive, the system entities E_1, ..., E_n cooperate to establish a cluster. By using *close*, each entity notifies all the entities of willing to close the cluster. If all the entities agree with it, the cluster is closed. By issuing *abort*, the entity can terminate the cluster unilaterally. The entities in the cluster are re-synchronized by issuing *reset*. A_i can issue a primitive *op* to C only if $op \in O_i$.

Suppose that A_1 is bound to a cluster C with a role $R_1 = \langle s_1, O_1 \rangle$. If $O_1 = \{receive\}$, A_1 can only receive messages sent in C while A_1 cannot send messages. If $O_1 = \{send, close\}$, A_1 can send messages and close the cluster.

From two roles R_i and R_j, *join* of R_i and R_j, $R_i \cap R_j$ is defined to be $\langle s, O \rangle$ where $s = class(R_i) \cap class(R_j)$ and $O = Op(R_i) \cap Op(R_j)$.

4.2 Mandatory access control

We would like to consider how communication primitives, i.e. *send* and *receive*, are related with information flow lattice. Each application entity A_i with role $R_i = \langle s_i, O_i \rangle$ sends and receives messages in the cluster C after C is established. The following mandatory access control is used when each A_i would like to send messages to A_j.

[Communication rule]

(1) A_i can *receive* messages sent by A_j if $receive \in O_i$ and $s_i \succeq s_j$.

(2) A_i can *send* messages to A_j if $send \in O_i$ and $s_i \preceq s_j$. □

Suppose that there are three application entities A_1, A_2, and A_3 whose classes are s_1, s_2, and s_3, respectively. Suppose that $s_1 \prec s_2 \prec s_3$. A_2 can send messages to A_3 and receive messages from A_1. A_1 can send messages to A_2 and A_3 but can receive messages neither from A_2 nor A_3. A_3 can receive messages from A_1 and A_2 but can send to neither A_1 nor A_2.

In the group communication, each A_i can send message to any subset of C, i.e. multiple entities. A_i can receive message sent to multiple entities. Hence, the communication rule has to be extended. A_i sends and receives message p in C by the following rule.

[Group communication rule]

(1) A_i can send message p to A_{i1}, ..., A_{in} if $send \in O_i$ and $s_i \preceq s_{i1} \cap ... \cap s_{im_i}$.

(2) A_i can receive message p sent by A_j if $receive \in O_i$, $A_i \in p.DST$ $(= \{A_{j1}, ..., A_{jm_j}\})$, and $s_j \preceq s_{j1} \cap ... \cap s_{jm_j}$ □

There are types of clusters on what information flow is supported. Let C be a cluster $\langle R_1, ..., R_n \rangle$ where $R_i = \langle s_i, O_i \rangle$. C is referred to as *balanced* iff every R_i has the same security class and $\{send, receive\} \subseteq O_i$. In the balanced cluster, every entity can send messages to every entity and receive messages from every entity.

4.3 Constrains

C is represented by a directed graph named a *cluster graph* where each node R_i shows a role R_i and there is a directed edge from R_i to R_j, i.e. $R_i \rightarrow R_j$ if $R_i \preceq R_j$. $R_i \rightarrow R_j$ is referred to as *supported* iff $send \in O_i$ and $receive \in O_j$. Even if $s_i \preceq s_j$, unless $R_i \rightarrow R_j$ is supported, A_i cannot deliver messages to A_j. R_i and R_j are referred to as *linked* (written as R_i — R_j) iff $R_i \rightarrow R_j$ are supported, R_j — R_i, or there is R_k such that R_i — R_k and R_k — R_j. C is referred to as *connected* iff for every pair of R_i and R_j, R_i and R_j are linked. If C is not connected, C is partitioned into disjoint subgroups. There is no supported link among any pair of subgroups while every subgroup is connected. This means that there is no way for any two subgroups to communicate with each other. Hence, C cannot be established if C is not connected.

[Example 2] In Figure 2, suppose that $O_1 = \{send\}$, $O_2 = \{receive, send\}$, and $O_3 = \{send\}$. $R_1 \rightarrow R_2$ is supported since $s_1 \preceq s_2$ and $send \in O_1$ and $receive \in O_2$. Neither $R_1 \rightarrow R_3$ nor $R_3 \rightarrow R_1$ is not supported since neither $receive \in O_1$ nor $receive \in O_2$. Figure 3 shows the cluster graph for Figure 2. □

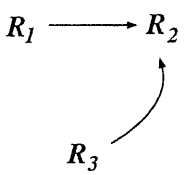

Figure 3: Cluster graph of Figure 2

Each application entity A_i is modeled as an object which has data structure D_i and a set of operations O_i for manipulating D_i. There are two kinds of operations in O_i from the information flow point of view. One kind of operation carries data. The other does not carry data. For example, *read* does not carry data which the response of read and *write* carry data. The former is *data* and the later is named *command*. Since the *command* message does not carry data, it can be sent to any entities in the cluster. On the other hand, the *data* message has to be sent to the entities along the supported edges in the cluster graph.

There are types of communication in the cluster. One type is a one-to-many communication like the client-server model. Let S and C_i be roles of server and client $(i = 1, ..., n)$, respectively. $class(S) \preceq class(C_i)$ if $Op(C_i) = \{receive\}$, i.e. retrieval. Every client can read information in the server. The cluster graph is star-structured where every client node is connected to the server node and any client nodes are not linked. In the

other type, each entity sends and receives message equally. Here, every role has the same security class. The cluster has to be balanced.

[Definition] $R_i = \langle s_i, O_i \rangle$ is referred to as *acceptable* for A_i if (1) $class(A_i) = s_i$ if $\{send, receive\} \subseteq O_i$, (2) $class(A_i) \preceq s_i$ if $send \in O_i$ but $receive \notin O_i$, and (3) $class(A_i) \succeq s_i$ if $receive \in O_i$ but $send \notin O_i$. □

If R_i is not acceptable for A_i, A_i cannot issue primitives in O_i to C.

4.4 Cluster establishment

We would like to discuss how system entities E_1, ..., E_n establish a roled cluster $C = \langle R_1, ..., R_n \rangle$ among application entities A_1, ..., A_n. Suppose that each A_i is supported by system entity E_i. There are two kinds of application entities, i.e. *active* and *passive* ones. Active entity A_i issues *active* open request primitive $aop(\langle A_1{:}R_1, ..., A_n{:}R_n \rangle)$ to E_i in order to send the open primitive to A_1, ..., A_n. Passive A_i issues *passive* open primitive pop $(\langle R_1, ..., R_n \rangle)$ and waits for open indication primitives from active entities. E_1, ..., E_n establish C by the following procedure. Hence, each E_i has variables r_1, ..., r_n to store the roles of the entities in C.

[Roled cluster establishment procedure]

(1) On receipt of active open primitive *aop* $(A_1{:}R_1, ..., A_n{:}R_n)$ at the system SAP S_i from A_i, E_i sends *Aopen* $(A_1{:}R_1, ..., A_n{:}R_n)$ to E_1, ..., E_n. Here, E_i is referred to as *active* and $r_i := R_i$ $(i = 1, ..., n)$.

(2) On receipt of passive open primitive pop $(R_1, ..., R_n)$ from A_i, E_i waits for the active open *Aopen* from active entities. Here, $r_i := R_i$ $(i = 1, ..., n)$.

(3) On receipt of *Aopen* $(A_1{:}R_1, ..., A_n{:}R_n)$ from E_j, $r_k = r_k \cap R_k$ $(k = 1, ..., n)$. If the cluster graph for $\langle r_1, ..., r_n \rangle$ is not connected or not acceptable for A_i, E_i broadcasts *Abort* and stops the procedure. Otherwise, E_i broadcasts *Popen* $(\langle r_1, ..., r_n \rangle)$ if E_i have not sent *Popen* and waits for *Aopen* or *Popen* from every entity.

(4) On receipt of *Aopen* or *Popen* from every entity, E_i broadcasts *Opened* $(\langle r_1, ..., r_n \rangle)$ in C.

(5) On receipt of *Opened* from every entity, C is established. □

A_i receives either *Aopen* or *Popen* $\langle R_{j1}, ..., R_{jn} \rangle$ from every A_j $(j = 1, ..., n)$. After receiving them, $R_i = \langle s_i, O_i \rangle$ is defined as $R_i = R_{1i} \cap ... \cap R_{ni}$, i.e. $s_i = s_{1i} \cap ... \cap s_{ni}$ and $O_i = O_{1i} \cap ... \cap O_{ni}$. If every A_i agrees with $\langle R_1, ..., R_n \rangle$, the cluster C is established as $\langle A_1{:}R_1, ..., A_n{:}R_n \rangle$. Each message p sent at S_i by A_i in C has a security class $class(p) = class(R_i) = s_i$.

5 Multi-Roled Entity

Each application entity A may join multiple clusters C_1, ..., C_m $(m \geq 2)$. Suppose that A is bound to C_i with role R_i $(i = 1, ..., m)$. A can communicate with entities in one cluster C_i with role R_i while communicating with another C_j with R_j. Such an entity as A is referred to as *multi-roled* because A plays multiple roles in multiple clusters. Multi-roled entity A can forward messages received in C_i to another C_j [Figure 4]. That is, more-sensitive information in C_i can be flown into less-sensitive entities in C_j. We have to control the information flow among the clusters.

Role R_i of A in C_i means that A can send or receive messages only in C_i. If A has a higher security class s_j in another C_j than s_i in C_i, the messages received in C_i are allowed to be sent to C_j. That is, A *has* a role with which A can *send* messages to the *dominating* clusters, *receive* from the *dominated* clusters, and both for *equivalent* clusters.

Suppose that A receives message p in C_i. A can send p in another C_j by the following rule.

[Multi-roled entity rule] On receipt of p in C_i, A can forward p to C_j if $s_i \preceq s_j$. □

[Example 3] Suppose that an application entity A is bound to three clusters C_i, C_j, and C_k with roles $R_i = \langle s_i, O_i \rangle$, $R_j = \langle s_j, O_j \rangle$, and $R_k = \langle s_k, O_k \rangle$, respectively. Suppose that $s_i \preceq s_j \preceq s_k$. If A receives message a in C_i, A can forward a to C_j and C_k because $s_i \preceq s_j$ and $s_i \preceq s_k$. A can forward message c received in C_j into C_k but not into C_j. A forwards c received in C_k neither into C_i nor C_j. □

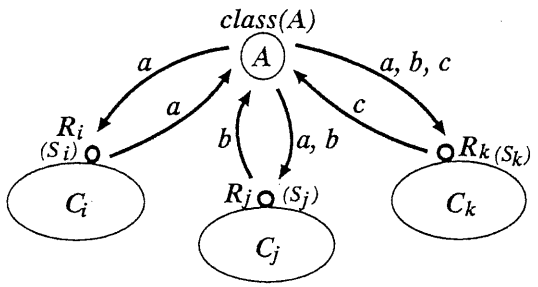

Figure 4: Multi-roled entity

6 Inter-Cluster Communication

In some case, entities in a cluster would like to send messages to another cluster. For example, suppose that there are two clusters, *database* cluster R and *teleconference* cluster T. R is composed

of redundant database servers. Users in T send update requests of the database to R. Here, information is flown into R from T. We would like to discuss the information flow among clusters.

Suppose that there are two clusters C_1 and C_2. Each C_i supports secure group communication and legal information flow for the entities in C_i. Suppose that entity A_i in C_i would like to forward message p to C_j. p has security class s_1 in C_1, and s_2 in C_2. For every pair of security classes s_1 and s_2, information of s_1 can be flown into s_2 iff $s_1 \rightarrow s_2$ according to the definition. There is security class s dominating s_1 in C_1, and dominated by s_2 in C_2, i.e. $s_1 \succ s \succ s_2$. Information of s can be flown from C_1 to C_2 if $s_1 \succ s \succ s_2$. If not, it cannot be flown. Let C_k denote a roled cluster $\langle A_{ik}{:}R_{ik}, ..., A_{kn_i}{:}R_{kn_k} \rangle$. Entity A_{ij} in C_i can send message p to C_j by the following rule.

[Inter-cluster information flow rule]

(1) $class(p)$ is changed into $s_{i1} \cap ... \cap s_{in_i}$, and

(2) p can be sent to C_j if $class(p) \preceq s_{j1} \cup ... \cup s_{jn_j}$. □

[Example 4] Figure 5 shows three clusters C_1, C_2, and C_3. Each C_i includes three application entities A_{i1}, A_{i2}, and A_{i3} (i = 1, 2, 3). In C_1, A_{11} and A_{13} plays role of security class s_1 and A_{12} plays role of s_2. In C_2, A_{21} and A_{23} have s_3 and A_{23} has s_3. In C_3, A_{31}, A_{32}, and A_{33} have s_1. Here, suppose that $s_1 \preceq s_2 \preceq s_3 \preceq s_4$. Suppose that A_1 would like to send message p to C_2. First, let $class(p)$ be $s_1 \cap s_2$, i.e. s_1. $class(R_{21}) \cup class(R_{22}) \cup class(R_{23}) = s_3 \cup s_1 \cup s_3 = s_4$. Since $s_1 \preceq s_4$, p is sent to C_2. □

In the inter-cluster information flow, each entity in a cluster C_i is allowed to send messages to another cluster C_j by using the *lub* of security classes in C_i, and to receive messages by using the *glb* of security classes in C_j. If not, they are rejected.

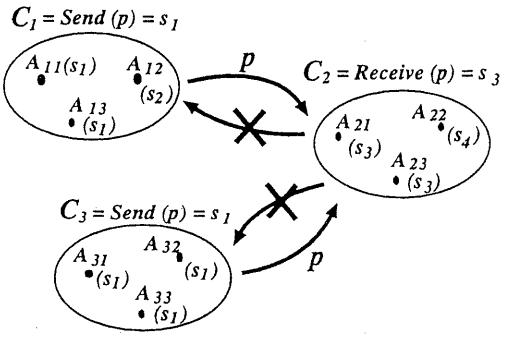

Figure 5: Inter-cluster information flow

7 Concluding Remarks

In this paper, we have discussed how to control the information flow in the cluster composed of multiple entities and the inter-cluster information flow on the basis of the security class. We have discussed the mandatory access control on the communication primitives, e.g. *send* and *receive*.

References

[1] Bell, D. E. and LaPadula, L. J., "Secure Computer Systems: Mathematical Foundations and Model," M74-244, The MITRE Corp., Bedford, Mass. (May 1973).

[2] Birman, K. P., Schiper, A., and Stephenson, P., "Lightweight Causal and Atomic Group Multicast," *ACM Trans. on Computer Systems*, Vol.9, No.3, 1991, pp.272-314.

[3] Denning, D. E., "Cryptography and Data Security," *Addison-Wesley*, 1982.

[4] Defense Communications Agency, "DDN Protocol Handbook," Vol.1–3, NIC 50004-50005, 1985.

[5] Ellis, C. A., Gibbs, S. J., and Rein, G. L., "Groupware," *Comm. ACM*, Vol.34, No.1, 1991, pp.38-58.

[6] International Standards Organization, "OSI – Connection Oriented Transport Protocol Specification," *ISO 8073*, 1986.

[7] Nakamura, A. and Takizawa, M., "Reliable Broadcast Protocol for Selectively Ordering PDUs," *Proc. of IEEE ICDCS-11* 1991, pp.239-246.

[8] Nakamura, A. and Takizawa, M., "Priority-Based Total and Semi-Total Ordering Broadcast Protocols," *Proc. of IEEE ICDCS-12*, 1992, pp.178-185.

[9] Nakamura, A. and Takizawa, M., "Causally Ordering Broadcast Protocol," to appear in *Proc. of IEEE ICDCS-14*, 1994.

[10] Ravi S. S., "Lattice-Based Access Control Models" *IEEE Computer*, Vol.26, No.11, 1993, pp. 9–19.

[11] Takizawa, M., "Design of Highly Reliable Broadcast Communication Protocol," *Proc. of IEEE COMPSAC'87*, 1987, pp.731-740.

[12] Takizawa, M. and Nakamura, A., "Partially Ordering Broadcast (PO) Protocol," *Proc. of IEEE INFOCOM'90*, 1990, pp.357-364.

[13] Takizawa, M. and Mita, H., "Secure Group Communication Protocol for Distributed Systems," *Proc. of IEEE COMPSAC'93* 1993, pp.159–165.