マルチメディア通信と分散処理 69－3
（1995．3．2）

グループ通信での情報流制御

三田 浩也　滝沢 誠
東京電機大学 理工学研究科 システム工学専攻
E-mail {mita, taki}@takilab.k.dendai.ac.jp

グループウエア等の分散型応用では、複数プロセス間でのグループ通信が要求される。グループ通信では、メッセージ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。グループ内の各プロセスは、グループ内の宛先プロセスにメッセージを選択的に送信する。ここで、メッセージを受信したプロセスが、このメッセージの宛先でないプロセスに転送するならば、宛先でないプロセスが結果として、メッセージを受信することになる。グループ通信では、こうした不正な情報流を制御する必要がある。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、グループ間での情報流を取り扱うための方式を論じる。

Information Flow Control in Group Communication

Hiroya Mita and Makoto Takizawa
Tokyo Denki University

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

1 Introduction

Distributed applications like groupware [7] are realized by the cooperation of multiple processes. Here, group communication among multiple communication processes in a group is required in addition to the conventional one-to-one communications supported by *TCP/IP* [4] and *OSI* protocols [9]. In the group communications [2,11,12,17,18], message sent by each process has to be delivered to either all the processes or none of them in the group, i.e. *atomic delivery* of messages. Each process also has to receive messages in some order [11,12,14]. For example, each process receives the same messages in the same order, i.e. *total order* [12], and receives messages in the *causal order* [2,13].

There are three kinds of group communication. A process sends messages to a group or multiple groups of processes. For example, a client sends messages to a group of multiple replicate servers where the messages are atomically delivered to the servers in some order. It is *group-oriented* communication or multicast discussed by [2]. [11–13, 17, 18, 16] discuss other type of group communication named *intra-group* communication. A collection of processes first establish a group and then send messages only to the processes in the group. The third type is *inter-group* communication [8]. Processes in a group send messages to processes in another group. In this paper, we would like to discuss the intra- and inter-group communications.

In addition, the *secure* group communication has to be supported. The following points have to be discussed to support the secure group communication.

(1) the authenticity and secrecy of intra-group communication.
(2) the secure information flow in the group, and
(3) the secure information flow among the group.

In the intra-group communication, it has to be guaranteed that each process receive messages from only the processes in the group, i.e. *authenticity*, and only the processes in the group send the massages in the group, i.e. *secrecy*. [19] discusses how only and all the proper processes 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 process would like to send messages to any subset in the group at any time, not necessarily to all the processes in the group. It is referred to as the *selective intra-group* communication [11]. After receiving message p from process E_i, if E_j forwards p to another E_k in the group, E_k can receive p from E_i, although E_i is not the destination of p. It is an 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 [5,14] is discussed to keep 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 is a lattice. [1] presents the *mandatory access control* model where access control and information flow are related. 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. Here, each process E_i has a *role* which denote what E_i can do in the group, i.e, set of primitives which E_i can issues. We would like to discuss how the primitives in the group communication 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. For example, clients may send data to a group of replicate database servers. Thus, information may be flown from one group to another. In this paper, we would like to discuss such *inter-group* information flow and give rules to keep the inter-group 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 secure intra-group information flow. In section 5, we discuss the secure inter-group information flow.

2 System Model

The communication system is composed of *application*, *system*, and *network* layers [Figure 1]. The network layer provides the system processes with the *reliable* high-speed communication [11–13, 17, 18]. The processes P_1, ..., P_n at the system layer can communicate with one another by using the network layer to provide the application processes A_1, ..., A_n with the secure group communication. A_i takes the service through system service access point (SAP) S_i supported by P_i. A *group* G of application processes A_1, ..., A_n is supported by the cooperation of P_1, ..., P_n, written as $G = \langle P_1, ..., P_n \rangle$. There are two kinds of group communications, i.e. intra- and inter-group communication. In the intra-group communication [11–13, 16–18], the processes send messages to only the processes in the group. For example, the members send messages to another in the tele-conference. In the inter-group communication [2,8,10], the processes send messages to a group or groups of processes. For example, a client sends messages to a group of printers to print out.

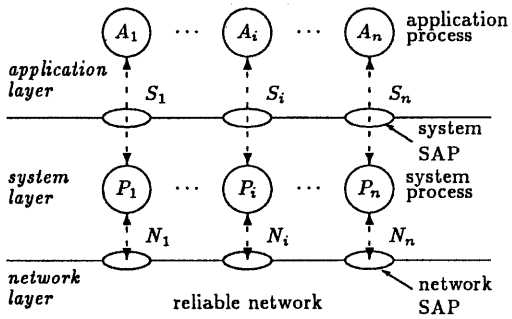

Figure 1: System model

In the group communication [2, 12, 18], each message p sent by system process P_i is delivered to all the processes in G. [11] discusses a selective group communication where P_i can send p to any subset of G at any time. Here, $p.DST$ denotes a set of destinations of p, and $p.DT$ shows data carried by p. In this paper, we assume that the network layer supports the selective group communication. A group G of A_1, ..., A_n is established by the cooperation of P_1, ..., P_n. Then, each A_i selectively sends messages to only the destinations in G by using the selective secure group communication. That is, A_i can send each message to only and all the destinations in G, i.e. *secrecy*, and can receive messages destined to A_i from only the processes in G, i.e. *authenticity*. [19] discusses how to realize the secure group communication [19] 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 unless $A_j \in p.DST$. In addition to realizing the secrecy and authenticity of the group communication, messages have to not be forwarded to the non-destination processes. In this paper, we would like to discuss how to provide the application processes with the secure information flow by using the reliable network.

3 Lattice-Based Model

We would like to present briefly a lattice-based model [5, 14] to deal with the information flow. Let S be a set of security classes. Every process belongs to one security class. Information in process has the security class of the process. 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 processes of s_2 iff $s_1 \rightarrow s_2$. That is, information in one process of security class s_1 can be stored in another process 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 [5] is described 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 $s_2 \preceq s_1$. s_1 and s_2 are not comparable ($s_1 \parallel s_2$) iff neither $s_1 \preceq s_2$ nor $s_2 \preceq s_1$.

Suppose that a group G supports application processes A_1, ..., A_n. Each A_i is supported by system process P_i ($i = 1, ..., n$). Let S be a set of security classes, 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 p to A_j if $class(A_i) \preceq class(A_j)$. That is, if $class(p) \preceq class(A_j)$, $p.DT$ can be stored in A_j on receipt of p. Otherwise, A_j cannot accept p.

[Example 1] Suppose that there are three application processes A_1, A_2, and A_3 supported by a group G, whose security classes are s_1, s_2, and s_1, respectively. Suppose that there is a *can-flow* relation $s_1 \rightarrow s_2$ but not $s_2 \rightarrow s_1$, i.e. $s_1 \prec s_2$ [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 $s_1 \prec s_2$. A_1 and A_3 can communicate with one another because $class(A_1) = class(A_3) = s_1$. □

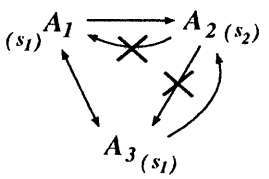

Figure 2: Information flow

[Definition] Let G be a group supporting application processes A_1, ..., A_n. The information flow in G is *legal* iff for every pair of A_i and A_j, A_i sends messages to A_j only if $class(A_i) \preceq class(A_j)$. □

The information flow in G is illegal if messages are delivered to A_j from A_i but not $class(A_i) \preceq class(A_j)$. In the secure group communication, every information flow among the application processes in G has to be kept legal.

The information systems are composed of subjects and objects. The subjects access the objects by using the primitives supported by the objects. The access control specifies which subjects could access what objects by what primitives. [1] discusses the *mandatory access control* based on the information flow lattice, where A can read information in object B if $class(A) \succeq class(B)$, and A can write information in B if $class(A) \preceq class(B)$. The former is the simple access property, and the latter is the *-property. At the application level, each application process A_i is modeled as objects, i.e. encapsulation of data structure and operations. The mandatory access rules are defined for the operations based on the information flow relation among the objects. A_i can issue operation *op* to A_j if the information in A_j is flown to A_i and $class(A_i) \succeq class(A_j)$. There are two kinds of flow, i.e. control and data. In the communication system, the processes can send and receive messages, and open and close the group.

4 Intra-group Communication

4.1 Roles

We would like to redefine a group G 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 processes can issue to G, 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 processes can issue to G, 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 processes A_1, ..., A_n establish G where each A_i is supported by system process P_i. Each A_i is referred to as *bound* to G with R_i if G is established by the cooperation of P_1, ..., P_n. It is written as $\langle A_1{:}R_1, ..., A_n{:}R_n \rangle$ named an *instance* of G which denotes a state of G being established. This means that each A_i plays a role R_i in G, i.e. A_i can issue only the primitives in $Op(R_i)$ to G. There are the following primitives for G, i.e. *send*, *receive*, *open*, *close*, *abort*, and *reset* primitives. *Open* primitive is issued to establish a group. On receipt of *open*, P_1, ..., P_n cooperate to establish a group. By using *close*, each process notifies all the processes of willing to close the group. If all the processes agree with it, the group is closed. By issuing *abort*, the process can terminate the group unilaterally. The processes in the group are re-synchronized by issuing *reset*.

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

For two roles R_i and R_j, $R_i \cap R_j$, *join* of R_i and R_j is defined to be $\langle s, O\rangle$ where $s = class(R_i) \cap class(R_j)$ and $O = Op(R_i) \cap Op(R_j)$. $R_i \cup R_j$, *union* of R_i and R_j is $\langle s, O\rangle$ where $s = class(R_i) \cup class(R_j)$ and $O = Op(R_i) \cup Op(R_j)$. $R_i \prec R_j$ if $send \in Op(R_i)$, $receive \in Op(R_j)$, and $class(R_i) \prec class(R_j)$. $R_i \equiv R_j$ if $\{send, receive\} \subseteq Op(R_i) \cap Op(R_j)$ and $class(R_i) = class(R_j)$. $class(R_i) \preceq class(R_j)$ if $class(R_i) \prec class(R_j)$ or $R_i \equiv R_j$.

4.2 Mandatory access control

We would like to consider how communication primitives are related with the information flow lattice. Each application process A_i with role $R_i = \langle s_i, O_i\rangle$ sends and receives messages in the group G after G is established. Here, let G be $\langle A_i : R_i, ..., A_n : R_n\rangle$. The following mandatory access control is used for sending and receiving messages in G.

[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 processes 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 multiple processes in G, and A_i can receive message sent to multiple processes. Hence, A_i sends and receives message p in G by the following rule.

[Group communication rule]

(1) A_i can send message p to A_{i1}, ..., $A_{i_{m_i}}$, i.e. $p.DST = \{A_{i_{11}}, ..., A_{i_{m_i}}\}$ 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 $s_j \preceq s_{j1} \cap ... \cap s_{jm_j}$ □

G is represented by a directed graph named a *group 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 $s_i \preceq s_j$. $R_i \rightarrow R_j$ 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. G is referred to as *connected* iff for every pair of R_i and R_j, R_i and R_j are linked. If G is not connected, G is partitioned into disjoint subgroups. This means that there is no way for any two subgroups to communicate with one another. Hence, G cannot be established if G is not connected.

[Example 2] In Figure 3, 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 supported since neither $receive \in O_1$ nor $receive \in O_2$. Figure 3 shows the group graph for Figure 2. □

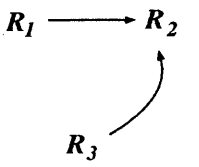

Figure 3: Group graph of Figure 2

There are types of communication in the group. One type is a one-to-many communication like the client-server model. Let S and G_i be roles of server and client $(i = 1, ..., n)$, respectively. $class(S) \preceq class(G_i)$ and $\{receive\} \in Op(G_i)$, i.e. retrieval, every client can read information in the server. The group graph is star-structured where every client node is connected to the server node and any client nodes are not directly linked. In the other type, each process sends and receives message equally. The group G is referred to as *balanced* iff every role R_i has the same security class and $\{send, receive\} \subseteq O_i$. In the balanced group, every process can send messages to every process and receive messages from every process. If G is not balanced, G is *unbalance*.

[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 acceptable, A_i can issue primitives to G. If R_i is not acceptable for A_i, A_i cannot issue primitives in O_i to G.

4.3 Group establishment

We would like to discuss how system processes P_1, ..., P_n establish a roled group $G = \langle A_1 : R_1, ..., A_n : R_n\rangle$. Suppose that each A_i is supported by system process P_i. There are two kinds of application processes, i.e. *active* and *passive* ones. Active process A_i issues *active* open primitive $aop(\langle A_1{:}R_1, ..., A_n{:}R_n\rangle)$ to P_i in order to send the open primitive to A_1, ..., A_n if $aopen \in Op(R_i)$. Passive A_i issues *passive* open primitive pop $(\langle R_1, ..., R_n\rangle)$ and waits for open primitives from the active processes. P_1, ..., P_n establish G by the following procedure. Here, each P_i has variables r_1, ..., r_n to store roles of the processes in G.

[Roled group establishment procedure]

(1) On receipt of active open primitive aop $(\langle A_1{:}R_1, ..., A_n{:}R_n\rangle)$ from A_i, P_i sends aop $(\langle A_1{:}R_1, ..., A_n{:}R_n\rangle)$ to P_1, ..., P_n. Here, P_i is referred to as *active* and $r_i := R_i$ $(i = 1, ..., n)$.

(2) On receipt of passive open primitive *pop* ($\langle R_1, ..., R_n \rangle$) from A_i, P_i waits for the active open *Aopen* primitives from the active processes. Here, $r_i := R_i$ $(i = 1, ..., n)$.

(3) On receipt of *aop* ($\langle A_1{:}R_1, ..., A_n{:}R_n \rangle$) from P_j, $r_k = r_k \cap R_k$ $(k = 1, ..., n)$. If the group graph for $\langle r_1, ..., r_n \rangle$ is not connected or not acceptable for A_i, P_i sends *Abort* to all processes and stops the procedure. Otherwise, P_i sends *pop* ($\langle r_1, ..., r_n \rangle$) to all the processes if P_i had not sent *Popen* and waits for *Aopen* or *Popen* from every process.

(4) On receipt of *Aopen* or *Popen* from every process, P_i sends *Opened* ($\langle r_1, ..., r_n \rangle$) in G.

(5) On receipt of *Opened* from every process, G 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 $R_{1i} \cap ... \cap R_{ni}$, i.e. $s_i = s_{1i} \cap ... \cap s_{ni}$ and $O_i = O_{1i} \cap ... \cap O_{ni}$ $(i = 1, ..., n)$. If every A_i agrees with $\langle R_1, ..., R_n \rangle$, i.e. R_i is acceptable for A_i and the group graph of G is connected, the group G is established as $\langle A_1{:}R_1, ..., A_n{:}R_n \rangle$.

Each message p sent at S_i by A_i in G has a security class $class(p) = class(R_i) = s_i$.

5 Inter-group Communication

5.1 Multi-roled process

Each application process A may join multiple groups G_1, ..., G_m $(m \geq 2)$. Suppose that A is bound to G_i with role R_i $(i = 1, ..., m)$. A can communicate with processes in one group G_i with role R_i while communicating with another G_j with R_j. A process A is referred to as *multi-roled* if A plays multiple roles in multiple groups. Multi-roled process A can forward messages received from G_i to another G_j [Figure 4]. That is, more-sensitive information in G_i can be flown into less-sensitive processes in G_j. We have to control the information flow among the groups.

Role R_i of A in G_i is concerned with whether A can send and receive messages in G_i. If A has a higher security class s_j in another G_j than s_i in G_i, the messages received in G_i are allowed to be sent to G_j.

Suppose that A receives message p in G_i. A can send p in another G_j by the following rule.

[Multi-roled process rule] On receipt of p in G_i, A can forward p to G_j if $s_i \preceq s_j$ and A can receive p in G_i. □

[Example 3] Suppose that an application process A is bound to three groups G_i, G_j, and G_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 G_i, A can forward a to G_j and G_k because $s_i \preceq s_j$ and $s_i \preceq s_k$. A can forward message c received in G_j into G_k but not into G_j. A forwards c received in G_k neither into G_i nor G_j. □

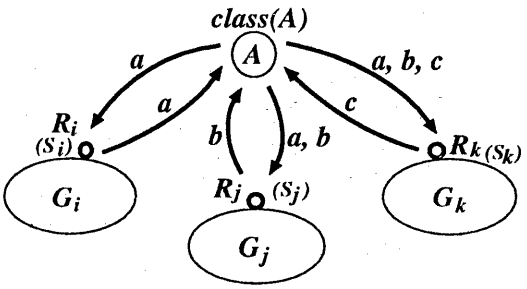

Figure 4: Multi-roled process

5.2 Inter-group information flow

In some case, processes in a group would like to send messages to another group. For example, suppose that there are two groups, *database* group R and *teleconference* group 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 groups.

Suppose that there are two groups G_1 and G_2. Each G_i supports secure group communication and legal information flow for the processes in G_i. Suppose that process A_i in G_i would like to forward message p to G_j. p has security class s_1 in G_1, and s_2 in G_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 G_1, and dominated by s_2 in G_2, i.e. $s_1 \succ s \succ s_2$. Information of s can be flown from G_1 to G_2 if $s_1 \succ s \succ s_2$. If not, it cannot be flown. Let G_k denote a roled group $\langle A_{ik}{:}R_{ik}, ..., A_{kn_i}{:}R_{kn_k} \rangle$. Process A_{ij} in G_i can send message p to G_j by the following rule.

[Inter-group information flow rule]

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

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

[Example 4] Figure 5 shows three groups G_1, G_2, and G_2. G_i includes three application processes A_{i1}, A_{i2}, and A_{i3} $(i = 1, 2, 3)$. In G_1, A_{11} and A_{13} play role of security class s_1 and A_{12} plays role of s_2. In G_2, A_{21} and A_{23} has role of s_3 and A_{23} of s_3. In G_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 G_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 G_2. □

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

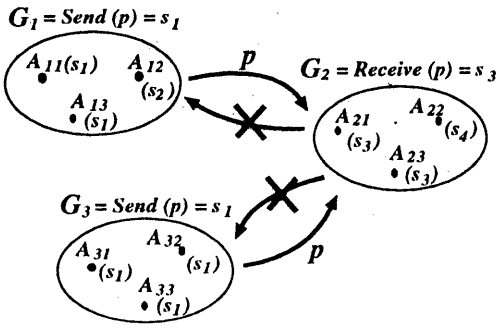

Figure 5: Inter-group information flow

6 Concluding Remarks

In this paper, we have discussed how to control the information flow in the group composed of multiple processes and the inter-group 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*.

Acknowledgements

We would like to thank Dr. Y. Murayama, Hirosima City Univ. for her helpful and useful discussions in this paper.

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] Chang, J. M. and Maxemchuk, N. F., "Reliable Broadcast Protocols," *ACM Trans. on Computer Systems*, Vol.2, No.3, 1984, pp.251-273.

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

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

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

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

[8] Higaki, H., "Group Communication Algorithm for Dynamically Updating in Distributed Systems," *Proc. of the IEEE ICPADS*, 1994, pp.56-62.

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

[10] Kaashoek, M. F. and Tanenbaum, A. S., "Group Communication in the Amoeba Distributed Operating System," *Proc. of the 11th IEEE ICDCS*, 1991, pp.222-230.

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

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

[13] Nakamura, A. and Takizawa, M., "Causally Ordering Broadcast Protocol," *Proc. of the IEEE ICDCS-14*, 1994., pp.48-55.

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

[15] Schneider, F. B., Gries, D., and Schlichting, R. D., "Fault-Tolerant Broadcasts," *Science of Computer Programming*, Vol.4, No.1, 1984, pp.1-15.

[16] Tachikawa, T. and Takizawa, M., "Selective Total Ordering Broadcast Protocol," *Proc. of the 2nd IEEE ICNP*, 1994, pp. 212–219.

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

[18] Takizawa, M. and Nakamura, A., "Partially Ordering Broadcast (PO) Protocol," *Proc. of the 9th IEEE INFOCOM*, 1990, pp.357-364.

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

[20] Tanenbaum, A. S., "Computer Networks (2nd ed.)," *Englewood Cliffs, NJ: Prentice-Hall*, 1989.