Role-Based Purpose-Oriented Access Control for Object-Oriented Systems

Tsunetake Ishida, Katsuya Tanaka, Hiroaki Higaki, and Makoto Takizawa
Tokyo Denki University
Email : {tsune, katsu, hig, taki}@takilab.k.dendai.ac.jp

Various kinds of distributed applications are developed by using object-oriented technologies like CORBA, which are widely used to realize the interoperability among the applications. In addition to realizing the interoperability, it is essential to make the distributed objects securely manipulated. The purpose-oriented access control model introduces a purpose concept to the access control model in the distributed objects. The purpose shows why a subject manipulates an object by invoking a method. In this paper, we discuss the purpose-oriented access control model in the object-oriented system and furthermore discuss how to incorporate role concepts in the purpose-oriented model.

1 Introduction

Various kinds of object-oriented systems [2] have been developed by using object-oriented languages like C++ and JAVA [9]. Object-oriented systems are composed of multiple objects. An object is an encapsulation of data and methods for manipulating the data. The objects are structured with *is-a* relations in the object-oriented systems while the objects are not related with *is-a* in the object-based systems. The Common Object Request Broker Architecture (CORBA) [12] is now getting a standard framework for realizing the interoperability among various kinds of distributed applications. In addition to realizing the interoperability, the applications are required to be secure, i.e. objects not only have to be protected from illegally manipulated but also have to be prevented illegal information flow [4, 14, 6] among objects.

In the basic access control model [10], an access rule is specified in a form (s, o, op) which means that a subject s is allowed to manipulate an object o by invoking a method op of the object o. A pair $\langle o, op \rangle$ is an *access right* granted to the subject s. Only requests which satisfy the access rules are accepted to be performed. Otherwise, the requests are rejected. The access control model implies the *confinement* problem [11], i.e. illegal information flow may occur among subjects and objects in the system. In order to make every information flow legal, the *lattice based* access control model [1, 4, 14] is proposed. The legal information flow is given by classifying objects and subjects and defining the *can-flow* relation [4] between classes of objects and subjects. In the *mandatory* model, the access rules are specified by an authorizer so that only the legal information flow occurs. For example, if a subject s reads an object o, information in o flows to s. Hence, the subject s is allowed to read the object o only if a *can-flow* relation from o to s is authorized. In the *discretionary* model [3, 5, 6], access rules are defined in a distributed manner while the mandatory access rules are specified only by the authorizer in a centralized manner. For example, a subject can grant other subjects an access rule granted to the subject in the relational database systems like Oracle [13] and Sybase [16]. In the *role-based* model [7, 15, 18], a *role* is defined to be a collection of access rights, which show a job function in the enterprise. The access rule is specified by granting subjects the roles only if the subjects can play the roles in the enterprise while each subject is granted an access right in the access control model. The role-based model is now being used in various kinds of applications since it is easier to grant access rights to subjects.

The traditional access control models discuss what subject can manipulate what object by what method. The authors [17, 19] newly propose a *purpose-oriented* model which takes into account a *purpose* concept why each subject manipulates objects in the object-based system. In the object-based system, methods are invoked in a nested manner. The purpose is modeled to be a method which invokes another method in the object-based system. It is critical to discuss how to specify access rules in the nested invocation of methods. One way is that a method op_1 of an object o_1 can invoke a method op_2 of an object o_2 if a subject which invokes op_1 is granted an access right $\langle o_2, op_2 \rangle$. Sybase [16] adopts the *ownership chain* mechanism where op_1 can invoke op_2 if the owner of o_2 is the same as o_1 even if s is not granted an access right $\langle o_2, op_2 \rangle$. It is not easy, possibly impossible to authorize access rules for huge number of autonomous objects and subjects. Another way is to speedily whether op_1 can invoke op_2 only if o_1 is granted an access right $\langle o_2, op_2 \rangle$. We take this *object pairwise approach.*

In addition, we discuss how to incorporate the role concepts into the purpose-oriented model in an object-oriented system where methods are invoked in the

nested manner.

In section 2, we present the role concept the object-oriented system. In section 3, we discuss the purpose-oriented access control model. In section 4, we discuss information flow.

2 System Model

2.1 Object-oriented system

Object-oriented systems are composed of objects. Objects are encapsulations of data and methods for manipulating the data. There are two kinds of objects; *classes* and *instances*. A class is defined to be a set of *attributes* and *methods*. An instance is a tuple of values, each of which is a value of an attribute in the class, with the methods of the class. A term "object" means an instance in most object-oriented environment like JAVA and C++.

A method of an object is invoked by sending a request message to the object. The method specified in the request message is performed on the object on receipt of the request. Then, the object sends the response back to the sender of the message. The method may further invoke methods in other objects. Thus, the invocations of the methods are *nested*.

A class can be derived from one or more classes. Here, suppose a class c_2 is derived from a class c_1. c_2 is referred to as a *subclass* of c_1. In turn, c_1 is a *supperclass* of c_2. The class c_2 inherits the attributes and methods of c_1. Here, c_2 *is-a* c_1. *Inheritance* provides means for building new classes from the existing classes. A class may *override* the definition of attributes and methods inherited from the supperclass.

Suppose that a method op_1 of an object o_1 invokes a method op_2 of an object o_2. There are types of invocations, i.e. *synchronous*, *asynchronous*, and *one-way* invocations. In the synchronous invocation, the method op_1 blocks after invoking op_2 until receiving the response of op_2. This is a well-known *remote procedure call* (RPC). In the asynchronous invocation, op_1 does not block and continues the computation after invoking op_2. However, op_1 eventually receives the response from op_2. This is similar to *fork* in Unix. In the one-way invocation, op_1 *neither* blocks after invoking op_2 nor receives the response from op_2. op_2 is performed independently of op_1. In the invocation of op_2 by op_1, the object o_1 plays a role of *subject* and o_2 plays a role of *object* in the access control model. In the nested invocation, the subject-object relation is relative. In this paper, we assume that every invocation is synchronous.

2.2 Roles

Each subject plays some *role* in an organization, e.g. professor, assistant, and student in a university. A role represents a job function that describes the authority and responsibility in the organization. Each person is assigned some role and then plays the role in the organization. In the *role-based access control* model [7,15,18], a *role* is modeled to be a set of *access rights*. An access right is given a pair of a method *op* and an object *o* which supports *op*, i.e. $\langle o, op \rangle$. That is, a role means what method can be performed on what object. A subject s is granted a role r only if s plays the role r in the organization. On the other hand, each access right is granted to subjects in the access control model. Here, a subject s is referred to as *bound* with the role r if r is granted to s. This means that s can perform a method *op* on an object o if $\langle o, op \rangle \in r$. For example, let us consider two roles *Professor* and *Student* in a university. In the university, professors give examinations to students. Then, the students write answers in the examination papers, and the professors mark the examination papers. An object *Paper* shows an examination paper. *Paper* supports methods *make* which writes questions in the paper, *write* which writes answers for the questions in the paper, and *mark* which marks the paper. Marks which the students obtain at the examinations are kept in record in another object *Record*. *Record* supports methods *record* which stores the marks of the papers for students, *look* which reads the record, and *change* which changes the marks stored in the record. Here, a role *Professor* is $\{\langle Paper, make\rangle, \langle Paper, mark\rangle, \langle Record, record\rangle, \langle Record, change\rangle, \langle Record, look\rangle\}$ and *Student* is $\{\langle Paper, write\rangle, \langle Record, look\rangle\}$. In the role-based model, a person who plays a role of *Professor* in the university is granted the role *Professor*. A student is granted the role *Student*, i. e. a collection of access rights. Thus, it is easier to grant subjects access rights than the access control model.

Some roles are *hierarchically* structured to represent logical authority and responsibility in an organisation. If a role r_i includes every access right of another role r_j, r_i is referred to as *higher than* r_j ($r_j \preceq r_i$). The relation "$\preceq$" is transitive. Here, the roles r_i and r_j are *uncomparable* if neither $r_j \preceq r_i$ nor $r_i \preceq r_j$. Here, let us consider an *Assistant* who can mark the examination papers and just look at the record, that is, $Assistant = \{ \langle Paper, mark\rangle, \langle Record, look\rangle \}$. Here, $Assistant \preceq Professor$ since $Professor \supset Assistant$. Professors cannot write answers in examination papers although they can make and mark the examination papers. However, *Student* can write papers. Therefore, *Student* is uncomparable with *Professor* and *Assistant*.

In a role-based model, each subject s can manipulate an object o by invoking a method *op* of o only if s is granted a role including an access right $\langle o, op \rangle$. If a subject s would like to exercise the authority of a role r with which s is bound, the subject s first establishes a *session* to the role r. Then, s can play a role of r, i.e. s can manipulate o by *op*. For example, a subject s can perform a method *mark* on an object *Paper* while a session between s and a role *Professor* or *Assistant* is established in Figure 1. The number of sessions which each subject can establish at the same time can be restricted to be one.

2.3 Authorization in nested invocations

Suppose that a subject s invokes a method op_1 on an object o_1 and then op_1 invokes a method op_2 on another object o_2. Here, suppose s is granted an access right $\langle o_1, op_1 \rangle$. In one way, only if s is granted an access right $\langle o_2, op_2 \rangle$, op_1 is allowed to invoke op_2 on behalf of s. In is cumbersome to specify access rights for each object. In Sybase [16], the ownership chain

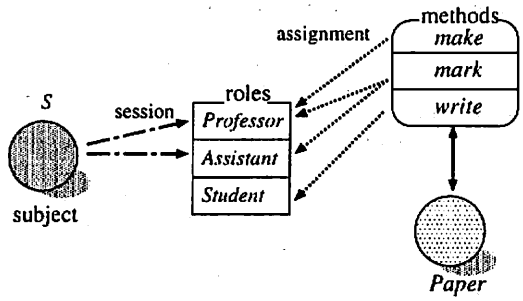

Figure 1: Roles.

method is adopted. Here, if the object o_2 has the same owner as the object o_1 and s is granted an access right $\langle o_1, op_1 \rangle$, op_1 can invoke op_2 even if s is not granted an access right $\langle o_2, op_2 \rangle$. Otherwise, op_1 is allowed to invoke op_2 only if s is granted the access right $\langle o_2, op_2 \rangle$. Suppose the response of op_2 carries some data derived from the object o_2. On receipt of the response, the object o_2 may store the data carried by the response in itself, e.g. the data is stored in the file of o_2 while o_2 continues to perform op_1 by using the response. This means, information in o_2 flows to o_1 through the invocation. The data may flow to other objects by further invocations. Thus, illegal information flow may occur by using the ownership chain method.

In this paper, we assume the system is composed of multiple autonomous objects, i.e. each object may have a different owner. Furthermore, it is difficult, maybe impossible for each autonomous object to grant access rights to subjects. In this paper, we take an object pairwise approach where access rules are authorized for a pair of autonomous objects o_i and o_j.

3 Purpose-Oriented Access Control

3.1 Purpose concept

The purpose-oriented model [17, 19] newly introduces a *purpose* concept to the access control model. A *purpose* shows why each subject s manipulates an object o by invoking a method op of o. In the object-based system, methods are invoked in the nested manner. Suppose a subject s invokes a method op_1 of an object o_1 and then op_1 invokes a method op_2 of an object o_2. In the purpose-oriented model, op_1 invoking a method op_2 of an object o_2 shows purpose for what o_1 manipulates o_2, while the access control model specifies whether or not o_1 can manipulate o_2 by invoking op_2. For example, suppose that a person s would like to withdraw money from a bank object B. In the access control model, the person s can withdraw money from B if an access rule $\langle s, B, withdraw \rangle$ is specified by the authorizer independently of purpose for what s spends the money. On the other hand, s can get money from the bank B for purpose of *house-keeping* but not for *drinking*. A *purpose-oriented* access rule $\langle s : house\text{-}keeping, B : withdraw \rangle$ is specified where a method *house-keeping* of the object s shows the purpose. Finally, the method op_1 of the object o_1 can invoke op_2 of o_2 only if the access rule $\langle o_1 : op_1, o_2 : op_2 \rangle$ is specified.

3.2 Hierarchical system

A role is specified in a collection of access rights in the role-based model [7, 15, 18]. We extend the purpose-oriented access control model to incorporate the role concept. In the object-based system, objects are related in the invocation relation. Let us consider *consumer* and *producer* objects as an example [Figure 2]. *Consumer* objects send requests to *Retailer* objects to purchase goods. *Retailer* objects sell the goods if they have. Otherwise, the *Retailer* objects order *Wholesaler* objects to send the goods. The *Wholesaler* objects obtain the goods from *Producer* objects. *Consumer* objects invoke methods supported by *Retailer* objects but invoke neither methods supported by *Wholesaler* nor *Producer* objects. The *Retailer* objects invoke methods supported by the *Wholesaler* objects but invoke neither *Producer* nor *Consumer* objects. The *Wholesaler* objects invoke methods supported by the *Producer* objects but invoke neither the *Retailer* nor *Consumer* objects. The *Producer* objects never invoke methods supported by the *Consumer*, *Retailer*, and *Wholesaler* objects. In this example, the objects can be classified into some levels. Objects at a level can invoke methods supported by objects of a lower level but cannot invoke objects of a higher level. Such a system is referred to as *hierarchical* system.

An object o_1 is *higher* than another object o_2 ($o_1 \succ o_2$) iff a method of o_1 invokes a method of o_2 or $o_1 \succ o_3 \succ o_2$ for some object o_3. Here, *Consumer* $\succ$ *Retailer* $\succ$ *Wholesaler* $\succ$ *Producer* in Figure 2. The objects are hierarchically structured in the system iff $o \succ o$ does not hold for every object o, i.e. $\succ$ is irreflexive. A pair of objects o_1 and o_2 are at the same level ($o_1 \equiv o_2$) iff neither $o_1 \succ o_2$ nor $o_2 \succ o_1$. For example, every pair of *Retailer* objects are at the same level. Objects at the level 0 are objects which are not invoked by any other objects. The *Consumer* objects are at level 0. Objects at the level i are objects which are invoked by objects at level $i-1$. In this paper, we assume that each object belongs to one level. That is, each object at level i invokes only methods of objects at level $i+1$. Objects which do not invoke methods of other objects are at the lowest level and named *primitive* objects. A *hierarchical* system is one where the objects are hierarchically structured. In this paper, we consider a system where objects are *hierarchically* structured in the invocation relation.

We consider roles in a hierarchical system. A role of a level i is a collection of access rights on the objects at the level i. Let R^i be a role of a level i which is $\{ \langle o^i, op \rangle \mid o^i$ is an object of the level i and op is a method of $o_i \}$. For example, let us consider a role. The *TravelAgent* and *Hotel* objects are at level i and $i+1$, respectively. The *BookTravel* and *Book* are the methods of *TravelAgent* and *Hotel*, respectively. *TravelTeller* is role at level i and, *TravelConsultant* and *HotelCustomer* are roles at level $i+1$ if *TravelTeller* includes an access right $\langle TravelAgent^i, BookTravel \rangle$ and, *TravelConsultant* and *HotelCustomer* include an access right $\langle Hotel^{i+1}, Book \rangle$, respectively. Here, The

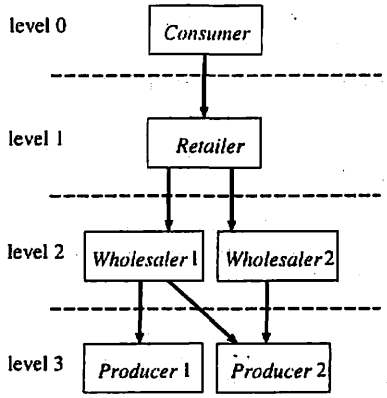

Figure 2: Hierarchical structure.

method *BookTravel* of the object $TravelAgent^i$ in the role *TravelTeller* can invoke the method *Book* of the object $Hotel^{i+1}$ in the role *TravelConsultant* and *HotelCustomer* [Figure 3].

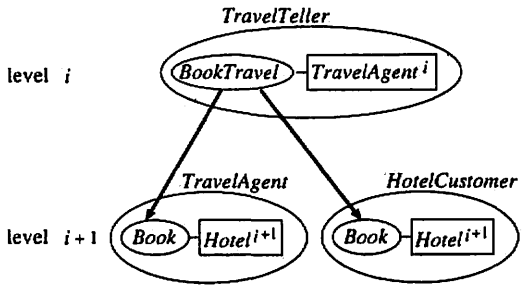

Figure 3: Hierarchical role.

Suppose that a method op_1 of an object o_1 invokes op_2 of o_2. Here, o_1 is at a level i and o_2 is at level $i+1$. We also suppose that an access right $\langle o_1, op_1 \rangle$ is in a role R_1. The method op_1 invokes a method op_2 of an object o_2 which is at level $i+1$. At level $i+1$, the access right $\langle o_2, op_2 \rangle$ is included in roles R_2^{i+1} and R_3^{i+1}. If an object o_1 is bounded with a role R_2^{i+1}, o_1 is allowed to invoke the method op_2 of the object o_2. This is a simple extension of the role-based model to the hierarchical object-based system. Let us consider the travel agent example shown in Figure 2. The method *BookTravel* of the *TravelAgent* object invokes a method *Book* of the *Hotel* object.

Each method op_i of an object o_i is granted a role $r_i = \{\langle o_{i1}, op_{i1} \rangle, \ldots, \langle o_{ih_i}, op_{ih_i} \rangle\}$. This means, the method op_i can invoke a method op_{ij} of an object o_{ij} (for $j = 1, \ldots, h_i$). In turn, op_{ij} may be granted a role $r_{ij} = \{\langle o_{ij1}, op_{ij1} \rangle, \ldots, \langle o_{ijh_{ij}}, op_{ijh_{ij}} \rangle\}$. op_{ij} can invoke a method op_{ijk} of o_{ijk} if op_{ij} is granted the role r_{ij}. An access rule has to show in what role the method op_i of the object o_i is bound with the role r_i. [Purpose-oriented role-based access (POR) rule] $\langle r : o_i : op_i, r_i \rangle$ means that a method op_i of an object o_i is invoked in a role r and op_i can invoke methods specified in a role r_i. □

[Example 1] Suppose that there are two roles *entertainment* and *house-keeping* including access right $\langle p,$ *drinking*$\rangle$ and $\langle p,$ *shopping*$\rangle$, respectively. A person p plays the roles in a community and manipulates the bank object b by authority of its role. If the method *drinking* of p is invoked in the role *entertainment*, p is allowed to withdraw money from the bank b. However, p is not allowed to do so if *drinking* of p is invoked in the role *house-keeping*. Thus, the access rule is specified in a form $\langle$*entertainment* : p : *drinking*, b : *withdraw*$\rangle$ where the method *drinking* shows the *purpose* of p. □

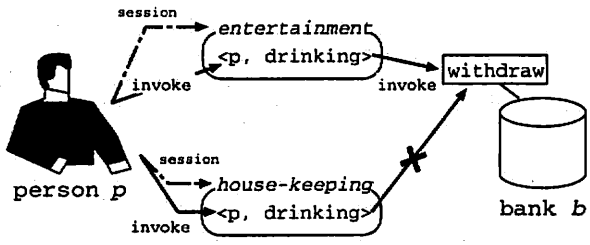

Figure 4: Purpose-oriented role-based access.

3.3 Class role

The object-oriented system is composed of classes and objects, i.e. instances of the classes. There are two kinds of access rights, *class* and *instance* access rights. A class access right is in a form $\langle c,$ *op*$\rangle$ where c is a class and *op* is a method of the class c. On the other hand, an instance access right is in a form $\langle o,$ *op*$\rangle$ where o is an object and *op* is the method of o.

There are two kinds of roles, i.e. class roles and instance roles. A class role r is defined in terms of methods and classes, i.e. $r = \{\langle c,$ *op*$\rangle$ where c is a class and *op* is a method of $c\}$. On the other hand, an instance role r' is defined in terms of methods and objects, i.e. $r' = \{\langle o,$ *op*$\rangle$ where o is an object and *op* is a method of $o\}$. r' is instantiated from the class role r. In the instance role r', o is an object which is instantiated from a class c.

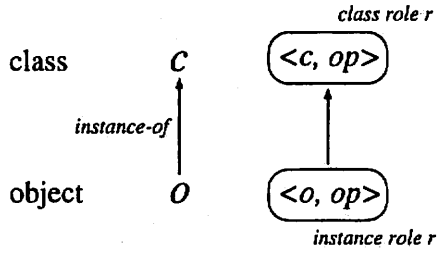

Figure 5: Class role and instance role.

For example, a class role *member* is defined as $member = \{\langle computer, use\rangle\}$ in Figure 6. A class role *member* is bound to a class *student*, i.e. $\langle student, member\rangle$. This means that the class *student* is authorized to access to the class *computer* by the method *use* and authority of the class role *member*. On the other hand, an object p is instantiated from a class *student* as an instance of *student*. PC_1 and PC_2 are also instantiated from a class *computer*. If p would manipulate PC_1 in the system, an instance role *member* is instantiated from a class role *member* to control the access between p and PC_1. An instance role *member* is associated to p. Even if PC_2 is an instance of *computer*, $\langle PC_2, use\rangle$ does not exist in the instance role *member* where p should not manipulate PC_2.

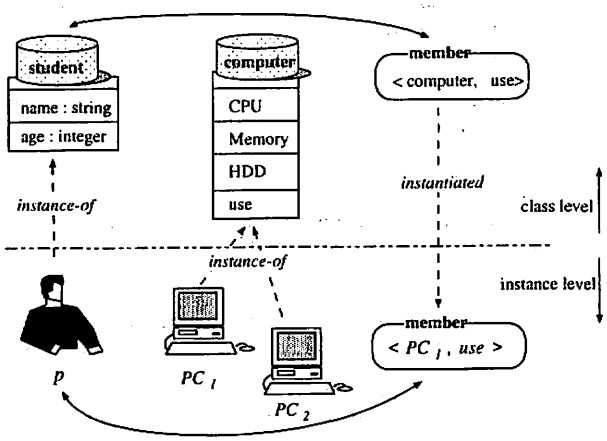

Figure 6: Instantiation of class and role.

Furthermore, there is an *is-a* relation in object-oriented systems. The *is-a* relation is defined among classes. We extend the role concept to conform to the *is-a* relation. Suppose that there are two classes c_1 and c_2. The class c_2 is defined as a specialization of the class c_1, i.e. c_2 *is-a* c_1. The access right $\langle c_2, op\rangle$ is automatically included in the role r where r is given as $\{\langle c_1, op\rangle\}$. This means that the access right of specialized class is given to the role when the role has an access right of its supperclass.

4 Information Flow Control

In the role-based access control model presented in the previous section, it is assured that subjects manipulate objects based on roles to which the subjects belong. However, illegal information flow among objects may occur. Because legal and illegal information flow among the objects are not discussed. For example, in Figure 7, suppose that a subject s_i invokes *write* on an object o_j after invoking *read* on o_i by the authority of a role r_i. This means that s_i may write data obtained from o_i to o_j. s_j can read data in o_i even if read access right is not authorize to a role r_j. This is the confinement problem pointed out in the basic access control model. In addition, a subject can have multiple roles in the role-based model even if they can play only one role at the same time. Suppose that a person A belongs to two roles *chief* and *clerk*. A person A obtains some information from *book* as a *clerk* and then stores the data derived from the information into book as a *chief*.

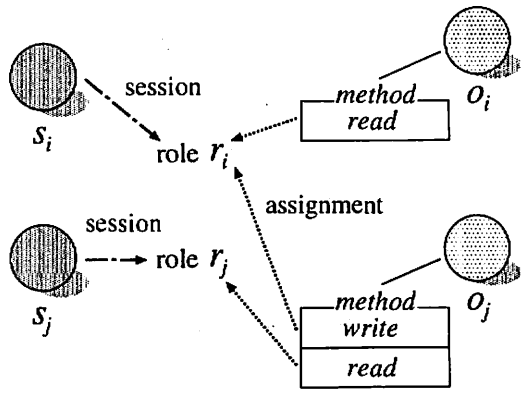

Figure 7: Illegal information flow.

We classify methods of objects with respect to the following points:

1. whether or not a value v_i of attribute a_i from an object o_i is output.
2. whether or not a value of a_i in o_i with input parameter is changed.

The methods are classified into four types in m_R, m_W, m_{RW}, and m_N. m_R means that the method outputs a value but does not change o_i. m_W means that the method does not output but changes o_i. The method m_{RW} outputs a value and changes o_i. The method m_N neither outputs a value nor changes o_i. For example, a method *count-up* is classified to be m_N because *count-up* changes the state of the object but does not need input parameter. *count-up* does not bring information into an object.

[Example 2] Let us consider a simple example about information flow between a pair of objects o_i and o_j in shown Figure 8. A subject s is now in a session with a role r_i. Here, s can invoke methods classified into m_R on o_i and m_{RW} on o_j by the authority of r_i, respectively. If s obtains information from o_i through m_R, s can invoke m_{RW} on o_j after the invocation of m_R on o_i. Because a set of roles on o_i which is authorized to execute methods classified into m_R is a subset of roles on o_j which is authorized to perform methods classified into m_R. □

5 Concluding Remarks

This paper has presented an access control model for distributed object-oriented systems with role concepts. Roles are higher level representation of access control models. We have defined a role to mean what method can be performed on which object. Furthermore, we have discussed how to control information flow to occur through roles.

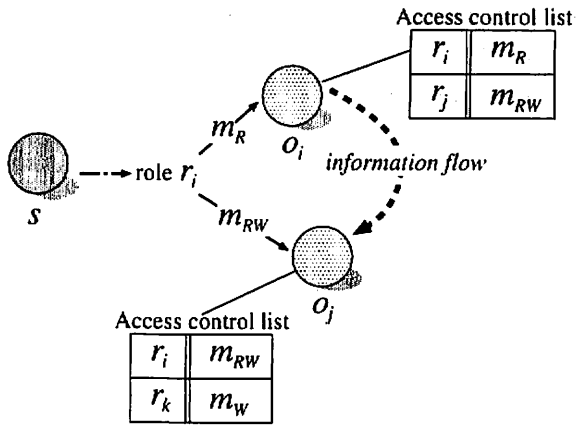

Figure 8: Information flow control.

References

[1] Bell, D. E. and LaPadula, L. J., "Secure Computer Systems: Mathematical Foundations and Model," *Mitre Corp. Report,* No. M74–244, *Bedford, Mass.,* 1975.

[2] Bertino, E. and Martino, L., "Object-Oriented Database Management Systems : Concepts and Issues," *IEEE Computer,* Vol. 24, No. 4, 1991, pp. 33–47.

[3] Castano, S., Fugini, M., Matella, G., and Samarati, P., "Database Security," Addison-Wesley, 1995.

[4] Denning, D. E., "A Lattice Model of Secure Information Flow," *CACM,* Vol. 19, No. 5, 1976, pp. 236–243.

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

[6] Ferrai, E., Samarati, P., Bertino, E., and Jajodia, S., "Providing Flexibility in Information Flow Control for Object-Oriented Systems," *Proc. of 1997 IEEE Symp. on Security and Privacy,* 1997, pp. 130–140.

[7] Ferraiolo, D. and Kuhn, R., "Role-Based Access Controls," *Proc. of 15th NIST-NCSC Nat'l Computer Security Conf.,* 1992, pp. 554–563.

[8] Harrison, M. A., Ruzzo, W. L., and Ullman, J. D., "Protection in Operating Systems," *CACM,* Vol. 19, No. 8, 1976, pp. 461–471.

[9] Gosling, J. and McGilton, H., "The Java Language Environment," Sun Microsystems, Inc, 1996.

[10] Lampson, B. W., "Protection," *Proc. of 5th Princeton Symp. on Information Sciences and Systems,* 1971, pp. 437–443. (also in *ACM Operating Systems Review,* Vol. 8, No. 1, 1974, pp. 18–24.)

[11] Lampson, B. W., "A Note on the Confinement Problem," *CACM,* Vol. 16, No. 10, 1973, pp. 613–615.

[12] Object Management Group Inc., " The Common Object Request Broker : Architecture and Specification," Rev. 2.1, 1997.

[13] Oracle, Corporation, "Oracle Server Administrator's Guide Release 8.0," 1997.

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

[15] Sandhu, R. S., Coyne, E. J., Feinstein, H. L., and Youman, C. E., "Role-Based Access Control Models," *IEEE Computer,* Vol. 29, No. 2, 1996, pp. 38–47.

[16] Sybase, Inc., "Sybase Adaptive Server Enterprise Security Administration," 1997.

[17] Tachikawa, T., Yasuda, M., and Takizawa, M., "A Purpose-oriented Access Control Model in Object-based Systems," *Trans. of IPSJ,* Vol. 38, No. 11, 1997, pp. 2362–2369.

[18] Tari, Z. and Chan, S. W., "A Role-Based Access Control for Intranet Security," *IEEE Internet Computing,* Vol. 1, No. 5, 1997, pp. 24–34.

[19] Yasuda, M., Higaki, H., and Takizawa, M., "A Purpose-Oriented Access Control Model for Information Flow Management," *Proceeding of 14th IFIP Int'l Information Security Conf. (SEC'98),* 1998, pp. 230–239.