Patterns for Session-Based Access Control

Eduardo B. Fernandez
Dept. of Computer Science and Engineerging
Florida Atlantic University
Boca Raton, FL 33431, USA
ed@cse.fau.edu

Günther Pernul
Department of Information Systems
Universität Regensburg
Universitätsstraße 31, Regensburg, Germany
guenther.pernul@wiwi.uni-regensburg.de

ABSTRACT

The concept of session, the context under which a user accesses resources is very important to apply access control. We present first the Controlled Access Session pattern for describing how sessions can limit the rights of a user. We then combine this pattern with two existing access control patterns. First we consider a pattern for Session-Based Role-Based Access Control, intended for organizations in which job functions form the basis for privilege assignments. Then, we present a Session-Based Attribute-Based Access Control pattern for organizations in which accesses are controlled based on values of user attributes and object properties. Since the general properties of those patterns have been described earlier we emphasize the additional effect of using sessions. The Controlled Access Session pattern can also be combined with other models of access control or used on its own.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Contructs and Features – *patterns.*
D.2.11 [**Software Engineering**]: Software Architectures – *patterns.*

General Terms

Security, Documentation, Algorithms, Management

Keywords

Access session, access control, attribute-based access control, session-based access control, security patterns

1. INTRODUCTION

It is important to develop systems where security has been considered at all stages of design, which not only satisfy their functional specifications but also satisfy security requirements. To do this we need to start with high-level models that represent the security policies of the institution. There are three models currently used by most systems: the access matrix, the Role-Based Access Control (RBAC) model, and the multilevel model.

One of the first security models was the access matrix. The basic access matrix [13] included the tuple {s,o,t}, where s indicates a subject or active entity, o is the protected object or resource, and t indicates the type of access permitted. [Har76] proved security properties of this model using the so-called HRU (Harrison-Ruzzo-Ullman) model. In that model users are allowed to delegate their rights (discretionary property, delegatable authorization), implying a tuple {s,o,t,f}, where f is a Boolean copy flag indicating if the right is allowed to be delegated or not. A predicate was added to the basic rule to allow content-based authorization [7], becoming {s,o,t,p,f}, where p is the predicate (the predicate could also include environment variables). Patterns for the basic rule and for the tuple {s,o,t,p,f} were given in [9][23]. The rule could also include the concept of Authorizer (a), becoming {a,s,o,t,p,f} [8] (Explicitly Granted Authorization). RBAC [22] can be considered a special interpretation of the basic authorization model, where subjects are roles instead of individual users. We presented two varieties of RBAC patterns in [9] and [23]. Subsequently, several variations and extensions of these models have appeared. We presented a variation called Metadata-Based Access Control, which later we renamed Attribute-Based Access Control (ABAC) [19][20].

ABAC can be seen in two ways:

- A specialization of the model {s,o,t,p}, where p is a predicate which depends on attribute values.
- A variant where s and o are defined by descriptors which depend on attribute values.

In this paper we present a general pattern for a Controlled Access Session as a building block and two patterns combining this pattern with specific access control models. The concept of session, the context under which a user accesses resources is very important to apply access control. We present first the Controlled Access Session pattern for describing how sessions can limit the rights of a user. We then combine this pattern with a pattern for Session-Based Role-Based Access Control, intended for organizations in which job functions form the basis for privilege assignments. Then, we present a Session-Based Attribute-Based Access Control pattern for organizations in which accesses are controlled based on values of user attributes and object properties. Since the general properties of those patterns have been described earlier we emphasize the additional effect of using sessions. The Controlled Access Session pattern can also be combined with other models of access control or used on its own. The pattern diagram of Figure 1 shows the relationships between these patterns. For example, adding a condition to Basic Authorization results in Content-Based Authorization, using the concept of session results in session-based models, and so on. Note that RBAC is, in general, not delegatable. All these patterns define authorization rules and they need a reference monitor for their enforcement; we don't show it in this diagram for simplicity (see [23] for the corresponding pattern). The double-lined patterns are the ones presented here. We assume the reader to know basic security concepts and these patterns are intended for system designers trying to add security to their designs.

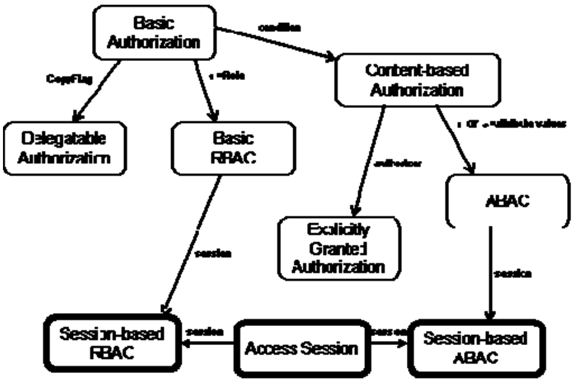

Figure 1. Relationships between access and control patterns

2. Controlled Access Session

Provide a context in which a subject (user, system) can access resources with different rights and without need to reauthenticate every time he accesses a new resource.

2.1 Example

Lisa is a secretary in a medical organization but sometimes she helps in the laboratory to perform patient tests. As a secretary she has access to patients' information such as name, address, SSN, etc. This is necessary so she can bill them and their insurance companies. In the lab she has access to anonymized patient test results. Combining the accesses provided by her two jobs in one window she can associate test results to names, which violates patient privacy.

2.2 Context

Any environment where we need to control access to computing resources and where users can be classified according to their jobs, groups, departments, assignments, or tasks.

2.3 Problem

A given user may be authorized to access a system because she needs to perform several functional activities. However, for a particular access only those privileges should be active which are necessary to perform the intended task. This is an application of the principle of least-privilege and necessary to prevent the user from misusing the system (intentionally, accidentally by performing an error, or without knowledge and tricked to do so, for example through a Trojan Horse attack). Additionally this would potentially restrict damage in case of session hijacking. A successfully attacking process would not have all privileges of a user available but only the active subset.

The following forces will affect the solution:

- Subjects may have many rights directly or indirectly through the execution contexts that they need for their tasks. Using all of them at one time may result in conflicts of interest and security violations. We need to restrict the use of those rights depending on the application or task the subject is performing.
- In the context of an interaction we can make the access to some functions implicit, thus facilitating the use of the system and preventing errors that may result in vulnerabilities. For example, some editors or other tools could be implicitly available in some sessions.
- It is not convenient to make subjects reauthenticate every time they request a new resource. Once the subject is authenticated, this condition should remain valid during the whole session.

2.4 Solution

Define a unit of interaction, a *session,* which has a limited lifetime, e.g. between login and logoff of a user or between the beginning and the end of a transaction. When a user logs on and after authentication, the session activates some execution contexts with only a subset of the authorizations she possesses. It should be the minimal subset which is needed for the user or transaction to perform the intended task. Only those rights are available within the session. A subject can be in several sessions at the same time; however, in every session only the necessary rights are active.

Structure

Figure 2 shows the class model of the Access Session pattern. Classes **Subject** and **Session** have the obvious meaning. The class **ExecutionContext** contains the set of active rights that the user may use within the session.

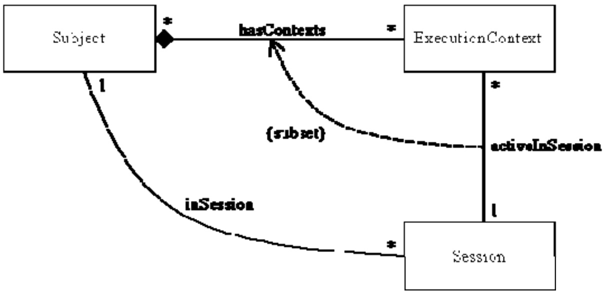

Figure 2. Class model for Access Session pattern

Figure 3 shows the use case Open (Activate) a session. A subject logs on and the logon interface authenticates it. The box with a double arrow indicates some authentication dialog or protocol. After the subject is authenticated, the interface creates a session object and returns a handle to the subject.

2.5 Implementation

Based on institution and application policies define which contexts (implying specific rights) should be used in each task and grant them to the corresponding subject. The rights should be selected using the least privilege principle and there should be no contexts with excessive rights, e.g. the administrator rights should be divided into smaller sets.

2.6 Example resolved

Lisa can log on a secretary or as a lab assistant but she cannot combine these activities in one session. Now she cannot relate results to patient names.

2.7 Known uses

- Session Access is part of the RBAC standard proposal by NIST which later has been adopted by the American National Standards Institute, International Committee for Information Technology Standards (ANSI/INCITS) as ANSI INCITS 359-2004 [10].
- Multics [Sum97] used execution contexts (based on projects) to limit access rights. Session Access is implemented in the security module CSAP [Dri03] of the Webocrat System in conjunction with an RBAC policy.
- Views in relational databases can be used to define sets of rights. Controlling the use of views by users can control their use of rights in sessions. This is done for example in Oracle and DB2, where SQL can be used to define restricted views [6].

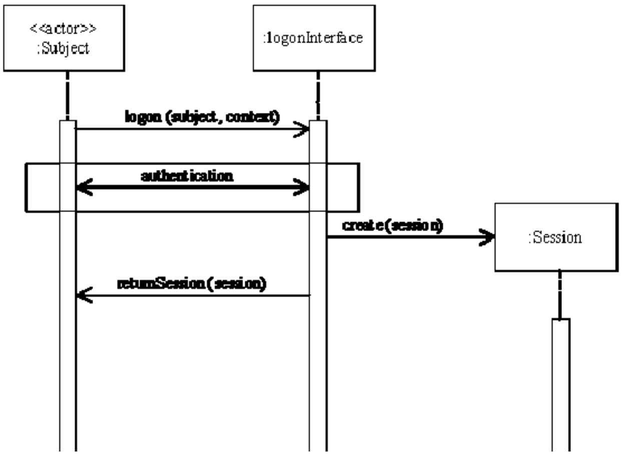

Figure 3. Sequence diagram for use case 'Open a session'

2.8 Consequences

This pattern has the following advantages:

- We can give to each context only the needed rights according to its function and we can invoke in a session only those contexts that are needed for a given task.
- We can exclude combinations of contexts that might result in possible access violations or conflicts of interest.
- Any functions can be made implicit in a session.
- Once a subject starts a session it doesn't have to be reauthenticated. Its status is kept by the session.

Possible disadvantages:

- If we need to apply fine-grained access, it might be inefficient to include many contexts to perform complex activities.
- Using sessions may be confusing to the users.

2.9 Related patterns

The Access Session pattern is used in the Session-Based RBAC and ABAC patterns, discussed later.

The Session pattern of [26] created a session object that defined a namespace to hold all the variables that need to be referenced by many objects. P. Sommerlad remade this pattern as a Security Session [23], intended to prevent a user to be reauthenticated every time he accesses a new object. A pattern with a similar objective to the previous one is Abstract Session [21]: When an object's services are invoked by clients, the server object may have to maintain state for each client. The server creates a session object that encapsulates state information for the client. The server returns a pointer to the session object. However, none of these patterns considers limitation of rights. Our pattern is an extension of those patterns, concentrating all its security functions and emphasizing the function of a session as a limiter of rights.

3. Session-Based Role-Based Access Control

Allow access to resources based on the role of the subject and limit the rights that can be applied at a given time based on the contexts (roles) defined by the access session.

3.1 Example

John is a developer in a project. He is also a project leader in another project. As a project leader he can evaluate the performance of the members of his project. He combines his two roles and adds several flattering evaluations about himself in the project where he is a developer. Later, his manager thinking that they came from the project leader of that project, gives John a big bonus.

3.2 Context

Any environment where we need to control access to computing resources, where users can be classified according to their jobs or their tasks, and where we assign rights to the roles needed to perform those tasks.

We assume the existence of a Session pattern that can be used for the solution.

3.3 Problem

In an organization a user may play several roles. However, for each access the user must act only within the authorizations of a single role (i.e. within the context of the role) or combinations of roles that do not violate institution policies. How do we force subjects to follow the policies of the institution when using their roles?

In addition to the forces defined for the Access Session pattern, the following forces apply to the solution:

- People in institutions have different needs for access to information, according to their functions. They may have several roles associated with specific functions or tasks.
- We want to help the institution to define precise access rights for its members so that the least privilege policy can be applied when they perform specific tasks..

- Users may have more than one role and we may want to enforce policies such as separation of duty, where a user cannot be in two or more specific roles in the same session.

3.4 Solution

A subject may have several roles. Each role collects the rights that a user can activate at a given moment (execution context), while a session controls the way of using roles and can enforce role exclusion at execution time.

Structure

The structure of the session-based RBAC is shown in the class diagram given in Figure 4. The class **Role** is an intermediary between subject and object holding all authorizations a user possesses while playing the role and acts here as an execution context. Within a **Session**, only a subset of the roles assigned to a **Subject** may be activated, i.e. only those necessary to perform the intended task. Roles may be composed according to a Composite pattern [11], where higher-level roles acquire (inherit) rights from the lower-level roles.

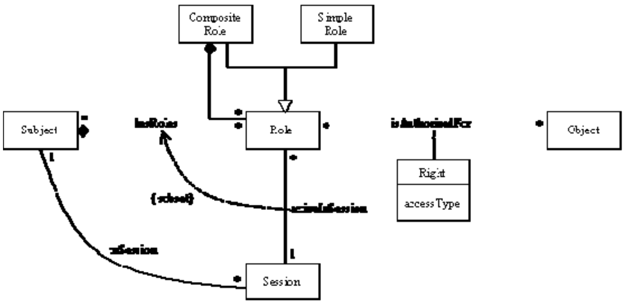

Figure 4. Class model for the Session-Based RBAC

Figure 5 shows a sequence diagram to request access to an object. A subject has already opened a session (See Figure 3) and he requests access to an object in a specific way (access type). The session uses the corresponding Reference Monitor, which in turn checks if the rights of the session roles allow the access. If so, the access is permitted.

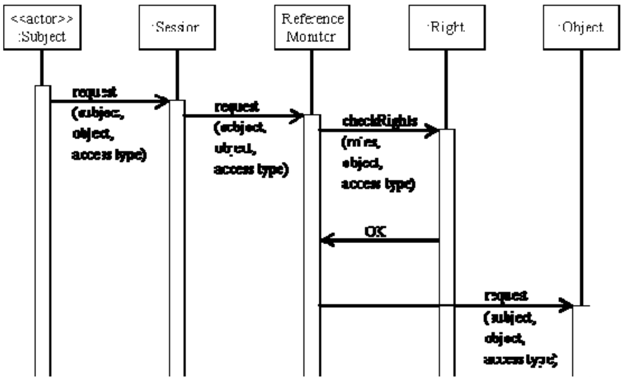
<<actor>> :Subject
:Session
Reference Monitor
:Right
:Object
request (subjec, object, access type)
request (subject, object, access type)
checkRights (roles, object, access type)
OK
request (subject, object, access type)

Figure 5. Sequence diagram to access an object

3.5 Implementation

See Section 5 for an example of a real implementation.

- Determine the roles the system should contain (role catalog), according to the user functions or tasks.
- Collect lists of incompatible roles and use these lists when a session is started (static constraints). These constraints can be defined using OCL or some other formal language as additions to the class diagram of the pattern.
- Determine the number of roles which may be active within a session (dynamic constraints).
- When a user opens a session she must declare what roles she intends to use and the system will open the corresponding session or refuse to do so in case of conflicts.

3.6 Example resolved

When John logs on the project where he is a developer he only gets the rights for a developer and cannot add evaluations. When he logs on in the project where he is a project leader he can only evaluate the members of his group. He cannot combine his role rights in the same session and now he only gets legitimate evaluations.

3.7 Known uses

The structure and dynamics of a session-based RBAC are implemented in the security module CSAP [5] of the Webocrat system. Webocrat is a portal supporting E-Democracy which was developed within the European Webocracy project (FP5-IST-1999-20364) between 2000-2003.

Views in relational databases can be used to define sets of rights. Controlling the use of views by roles can control the use of rights in sessions. In both Oracle and DB2 SQL can be used to define restricted views based on roles [6].

3.8 Consequences

In addition to the advantages mentioned for the Access Session pattern, other advantages of this pattern are:

- Sessions may include all needed roles for those subjects authorized for some task.

- Users can activate more than one session at a time for functional flexibility (some tasks may require multiple roles).
- Fine-grained rights can be assigned to roles to enforce a need-to-know policy.
- When a session is open, we can exclude roles that violate institution policies.

Possible disadvantages include:

- Additional conceptual complexity to define which roles can be used together and which should be mutually exclusive.
- User confusion if they have to use several roles to perform their work.

3.9 Related Patterns

This pattern is a combination of the Session pattern described earlier and the RBAC pattern [23]. As indicated earlier, structuring of roles can be represented by a Composite pattern. A Reference Monitor pattern is needed to enforce the use of rights during execution.

4. Session-Based Attribute-Based Authorization

Allow access to resources based on the attributes of the subjects and the properties of the objects but limit the rights that can be applied at a given time based on the context defined by the access session.

4.1 Example

Meili is a teenager who likes movies and subscribes to several movie services through the Internet. She logs in a central portal where she can reach a variety of movies. Sometimes she gets movies that she finds offensive or inappropriate (pornographic, racist, plain stupid). She doesn’t have much time to read details about the movies in advance and some of them don’t even have good descriptions so reading about the movies is not a good approach. She would like some kind of filter according to her characteristics and her preferences. Also the portal may be breaking the law in making available to her some of these movies.

4.2 Context

Dynamic systems supporting a large set of objects and subjects in which the structure of the subjects changes rapidly, such as web-based information systems, e-government and e-business portals. In this environment there is the need to control access to computing resources and the subjects may not be preregistered. We want to give access to resources based on characteristics of the subjects such as groups to which they belong, company for which they work, biological characteristics such as age or sex, or on characteristics of the objects, such as type of object, filtering rules, or payment requirements.

4.3 Problem

As indicated access may depend on the age or other attributes of a user. In this case, privilege assignments to the user cannot be done statically by a security administrator but automatically by the system based on the value of some of the attributes, e.g. “DateOfBirth” . As the user gets older or changes functions his authorization state changes automatically. Access rights might even depend on an external attribute, such as “physical location” of a user in a mobile environment. In this case the authorization state changes automatically when the user moves around. At the object’s side, metadata such as the scope of a document, or the MPAA rating of a movie are examples of properties. All these constraints can be applied through predicates in the rules [8], but it is difficult to have a variety of prepackaged rules for the typical cases.

The solution is constrained by the following forces:

- We need to limit the rights of subjects that are in a variety of groups or roles, or have special characteristics. Unrestricted access might allow policy or law violations.
- This control should not imply an extra burden for the security administrator or security vulnerabilities may appear through administration errors.
- This control should not imply a significant performance overhead, or the system may not be practical to use. .
- The environment is very dynamic and changes should be easy to make. Otherwise, the users will get annoyed and leave the system.

4.4 Solution

Access rights are based on the comparison of values of selected attributes of subjects and properties of objects (so called subject and object descriptors). In this pattern descriptors are a construct to somehow “group” objects and subjects dynamically, not explicitly by an administrator but implicitly by their attribute or property values. This grouping may result in unpredictable sets of rights that may violate security policies. A session delimits the rights that can be applied at a given moment; that is, the subject attributes define a context for access rights.

Structure

Figure 6 shows the class diagram for the solution. A **Subject Descriptor** is formed by applying **Qualifiers** (>, +,…) to **Attribute Values** to define constraints such as ‘age > 15’ . A **Session** selects some specific attribute values as execution context that defines the **Subject** descriptor at this moment. Similarly, objects are defined based on the values of selected attributes.

4.5 Implementation

See Section 5 for an example of a real implementation.

1) Select an appropriate package to convey the subject’s credentials including attributes. Examples would be attribute certificates [15][17] or Kerberos tickets.

2) Select an implementation to express the object’s attributes. Candidates could be standards on meta-data resource discovery, such as the Dublin Core Metadata Initiative [DCM].

3) Define an enforcement mechanism for the rights defined in contexts. See for example [2].

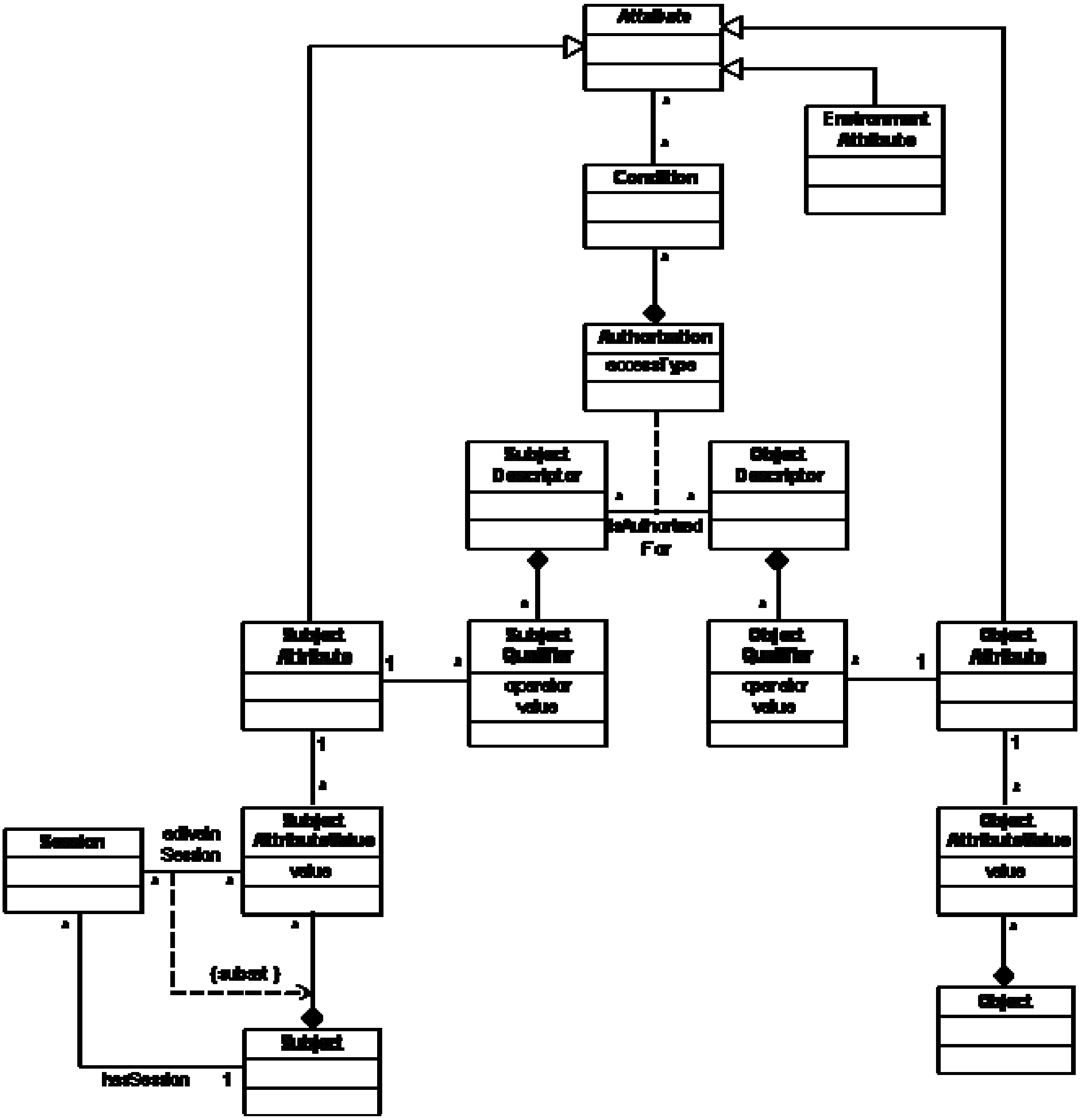

Figure 6. Class model for the Session-Based ABAC pattern

4.6 Example resolved

The portal implemented an ABAC model. Now when Meili opens a session she is given access to contexts with sets of preselected movies according to her preferences and restricted according to legal aspects and to the services she has paid for.

4.7 Known uses

Session-based ABAC is implemented as an alternative to RBAC in the security module CSAP [5] of the Webocrat system. A similar pattern is also used in the authorization system of the .NET component framework [14] and in AAIs (authentication and authorization infrastructures), such as Permis [1] and Shibboleth [24].

The XML standard XACML [4][16] uses attributes of subjects and objects for the specification of access control policies. As shown in the UCON_{ABC} [18], ABAC may also have potential for digital rights management.

4.8 Consequences

The advantages of this pattern include:

- The rights of subjects that belong to a variety of groups, roles, or have special attributes can be limited by restricting them to use specific contexts selected by sessions.
- This control does not imply an extra burden for the security administrator because the contexts can be defined by application designers according to application policies.
- This control does not imply a significant performance overhead because changing from one context to another just means changing a set of rights.
- Changes in access restrictions can be easily accommodated by defining new contexts or deleting existing contexts.

Possible disadvantages are:

- Higher complexity. Although the contexts are defined by others, it is hard for administrators to know who has access to what.
- There might still be some performance overhead if we need to switch often between contexts.

4.9 Related Patterns

Figure 1 shows the relationship of this pattern to other access control patterns. As indicated credentials such as certificates are frequently used to request access [15].

5. Using session-based access control as a service

In this section we show by means of two sequence diagrams how the patterns described above can be embedded into a general authentication, authorization and access control service. Such a service can be called by any application or process having the need to authenticate the users and to provide session-based access control. In the following it is assumed that the service provides both session-based RBAC and session-based ABAC and the client application requesting the service must chose between the two.

Figure 7 shows a sequence diagram for the interaction of a requesting client process and the session-based access control service. In order to hide the complexity of the subsystems, in the sequence diagram we use the Facade pattern [11] as a uniform interface for calling applications.

In order to be able to access a resource, a valid session object must be requested by the calling application (or user process). This starts with some sort of initialization process during which the client application first requests from the authentication facade of the security service an authentication service. In the example of Figure 7, a password service is returned but also other services may be available. Second is the request for an authorization service. In the example, RBAC is returned, and the initialization phase is finished. Next is the actual user authentication, role selection and the session establishment. During user authentication the client application provides to the password service <user-id, pwd>. The password service interacts with a userDM and in case of successful log-in a user object is created and a reference to the object (aUser) is returned to the calling client application.

A valid session can only be established in the case the user application activates at least one role from the set of possible roles for the user. This starts by calling the method getAssignedRoles of the user object. In case of a valid userID all available roles for a particular user are determined and returned by the role data module (RoleDM) and for each role a transient role object is created by the RBAC service. Next from the set of possible roles the user selects a subset and the RBAC service calls the corresponding method to activate the roles.

At this stage the user object is authenticated and has a set of active roles assigned. These are the only prerequisites for establishing a session. After receiving the request the session service creates a valid session object for which the session-id is returned as a reference for the calling client process. Under a valid session-id the client may act under the context of the session by using the privileges of the selected roles.

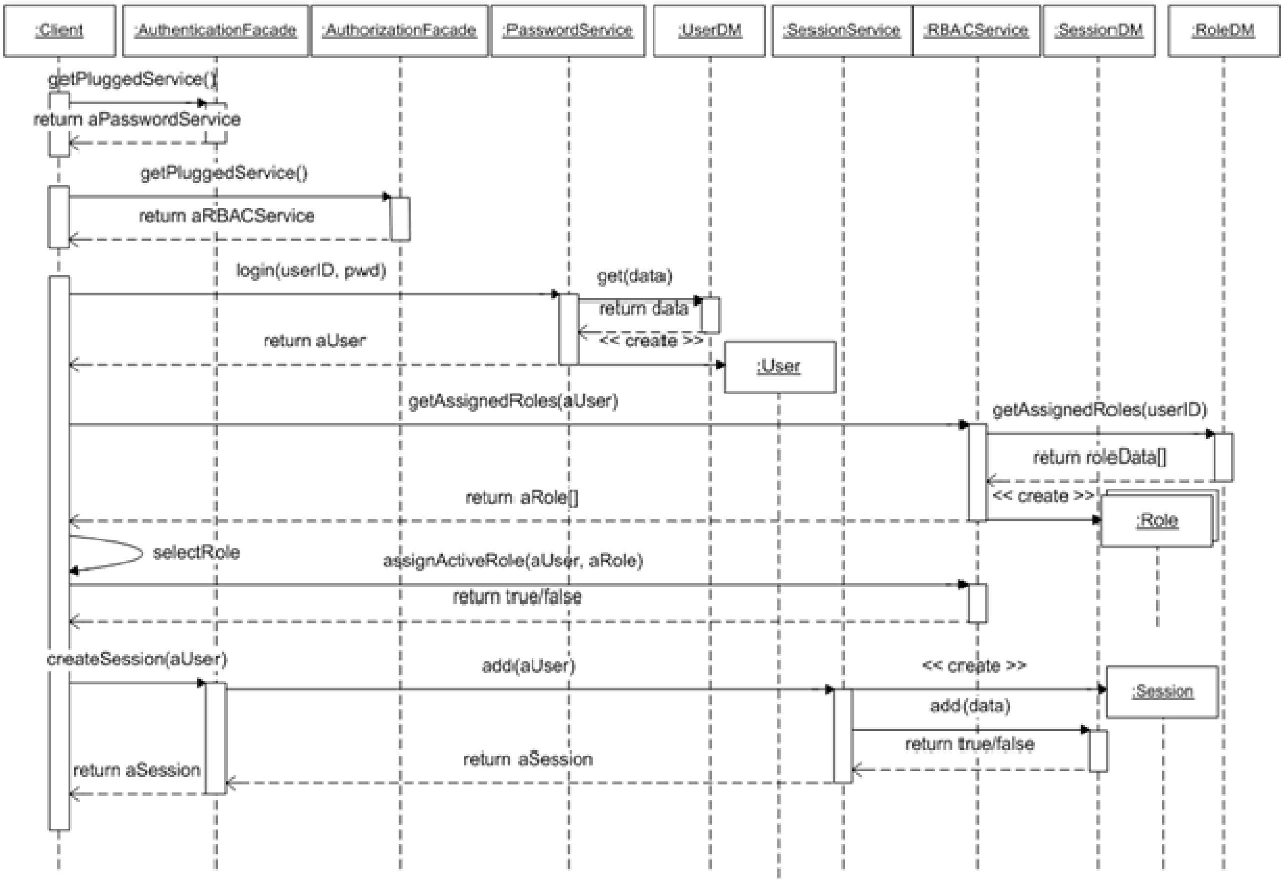

Figure 7. Session establishment

Figure 8 shows an attempt of a client process to access a resource within a valid session. The process starts with calling the method checkAccess with parameters session-id, object-id, operation, i.e. a request of a user wishing to access a certain object by using a predefined operation and this all within the context of an established session. First, the validity of the session is checked, then the session object is used by the RBACService in order to get the user's active roles within this session. Next, the user's permissions are determined by retrieving all the permissions assigned to the active roles. Finally, the RBACService checks whether there is a permission for the tuple <object, operation>. In the case there is one, the access will be granted, otherwise denied.

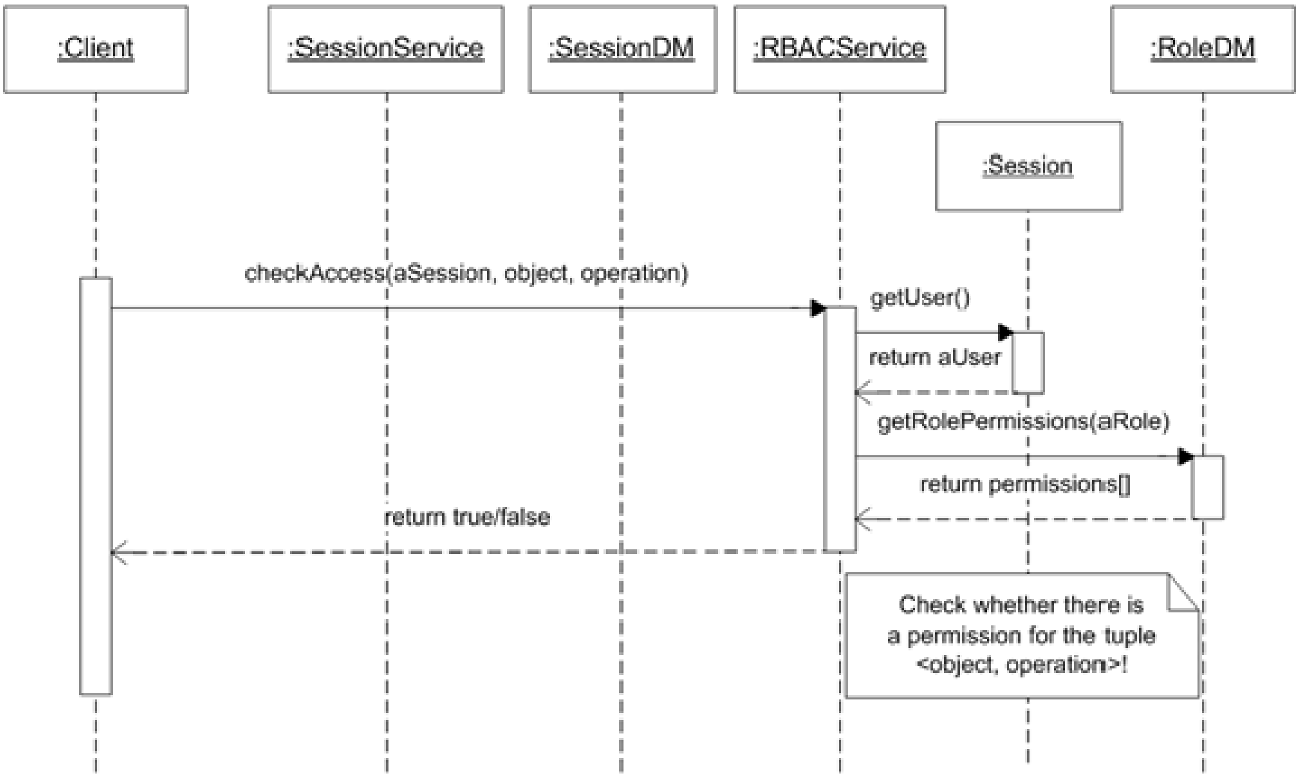
:Client
:SessionService
:SessionDM
:RBACService
:RoleDM
:Session
checkAccess(aSession, object, operation)
getUser()
return aUser
getRolePermissions(aRole)
return permissions[]
return true/false
Check whether there is a permission for the tuple <object, operation>!

Figure 8. Permission approval

6. Conclusion

We have shown patterns to describe the effect of sessions on access control models. We presented first the Access Session, which describes the basic concept of session as a limiter of rights. We then combined this pattern with the patterns of two access control models to show its effect on them. Finally we showed an example of a system using the last two patterns as a way to illustrate a real implementation.

Acknowledgements

We thank our shepherd Michael Weiss for his valuable comments that contributed to improve this paper. The FAU Secure Systems Research Group provided useful improvements and corrections. The group at the PLoP 2006 Writers Workshop provided very useful suggestions. The work of E. Fernandez was supported by a Federal Earmark grant from DISA, administered by Pragmatics, Inc. The work of G. Pernul was partly supported by the European Commission DG INFSO under the IST program, Webocracy, contract No. IST-20364.

References

[1] Chadwick, D. W., and Otenko, A. 2003. The PERMIS X.509 role based privilege management infrastructure. Future Generation Computer Systems. 19, 2 (2003), 277-289.

[2] Corradi, A., Montanari, R., and Tibaldi, D. 2004. Context-based access control management in ubiquitous environments. In Proceedings of the third IEEE Int. Symp. On Network Computing and Applics. NCA'04.

[3] The Dublin Core Metadata Initiative. DOI= http://www.dublincore.org.

[4] Delessy, N., Fernandez, E. B., and Sorgente, T. 2005. Patterns for the eXtensible Access Control Markup Language. In Proceedings of the Pattern Languages of Programs Conference (Allerton Park, IL, September 2005). PLoP 2005.

[5] Dridi, F., Fischer, M., and Pernul, G. 2003. CSAP -- an adaptable security module for the e-government system Webocrat. In Proceedings of the 18th IFIP International Information Security Conference (Athens, Greece, 26-28 May 2003). SEC 2003.

[6] Elmasri, R., and Navathe, S. 2003. Fundamentals of database systems (4th Ed.), Addison-Wesley.

[7] Fernandez, E. B., Summers, R. C., and Coleman, C. B. 1975. An authorization model for a shared data base. In Proceedings of the 1975 SIGMOD International Conference (New York, 23-31).

[8] Fernandez, E. B., Summers, R. C., and Wood, C. 1981. Database Security and Integrity. Addison-Wesley, Reading, Massachusetts, Systems Programming Series.

[9] Fernandez, E. B., and Pan, R. 2001. A pattern language for security models. In Proceedings of PLoP 2001.

[10] Ferraiolo, D., Sandhu, R., Gavrila, S., Kuhn, D.R. and Chandramouli, R. 2001. Proposed NIST standard for Role-Based Access Control. ACM Trans. on Information and System Security. 4, 3 (August 2001), 224-274.

[11] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. 1994. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Boston, Massachusetts.

[12] Harrison, M., Ruzzo, W., and Ullman, J. 1976. Protection in Operating Systems. Comm. of the ACM. 19, 8 (August 1976).

[13] Lampson, B. W. 1971. Protection. In Proceedings of the 5th Princeton Conference on Information Sciences and Systems. (Princeton, 1971).

[14] LaMacchia, B.A., Lange, S., Lyons, M., Martin, R., and Price, K.T. 2002. NET framework security. Addison-Wesley.

[15] Morrison, P., and Fernandez, E.B. 2006. The Credential Pattern. Pattern Languages of Programs conference (Portland, Oregon, USA, 2006). PLoP 2006.

[16] eXtensible Access Control Markup Language (XACML), Version 1.1. OASIS Community Specification, August 2003. DOI= http://www.oasis-open.org/committees/xacml/

[17] Oppliger, R., Pernul, G., and Strauss, C. 2000. Using Attribute Certificates to implement Role-Based Authorization and Access Control. In Proceedings of the 4th Conference on "Sicherheit in Informationssystemen" (Zürich, Switzerland, October 5 - 6, 2000). SIS 2000. vdf Hochschulverlag.

[18] Park, J., and Sandhu, R. 2004. The UCONABC usage control model, ACM Transactions on Information Systems Security. 7, 1 (February 2004), 128-174.

[19] Priebe, T., Fernandez, E. B., Mehlau, J. I., and Pernul, G.2004. A pattern system for access control. In Research Directions in Data and Applications Security XVIII, C. Farkas, and P. Samarati, Eds. Proceedings of the 18th. Annual IFIP WG 11.3 Working Conference on Data and Applications Security (Sitges, Spain, July 25-28).

[20] Priebe, T., Dobmeier, W., Muschall, B., and Pernul, G. 2005. ABAC – Ein Referenzmodell für attributbasierte Zugriffskontrolle. In Proceedings Sicherheit 2005, 2. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik. (Regensburg, April 2005).

[21] Pryce, N. 1997. Abstract session: An object structural pattern. In Proceedings of the 4th Conference of Pattern Languages of Programs. PLoP'97. Also in chapter 7 in Pattern Languages of Program Design, vol. 4, N. Harrison, B. Foote, and H. Rohnert, Eds. Addison-Wesley.

[22] Sandhu, R., Coyne, E. J., Feinstein, H. L., and Youman, C. E. 1996. Role-based access control models. IEEE Computer. 29, 2 (February 1996), 38-47.

[23] Schumacher, M., Fernandez, E.B., Hybertson, D., Buschmann, F., and Sommerlad, P. 2006. Security Patterns: Integrating security and systems engineering. J. Wiley & Sons.

[24] Shibboleth Project. DOI= http://shibboleth.internet2.edu

[25] Summers, R. C. 1997. Secure Computing: Threats and Safeguards. McGraw-Hill.

[26] Yoder, J., and Barcalow, J. 2000. Architectural Patterns for Enabling Application Security. In Proceedings of the 4th Conference of Pattern Languages of Programs. PLoP'97. Also in chapter 15 in Pattern Languages of Program Design, vol. 4, N. Harrison, B. Foote, and H. Rohnert, Eds. Addison-Wesley.