Patterns for fine-grain access-controlled business objects

RUSS RUBIS, FLORIDA ATLANTIC UNIVERSITY

DR IONUT CARDEI, FLORIDA ATLANTIC UNIVERSITY

ABSTRACT

Businesses constantly consume, generate and share data. Without data, a typical business process would come to a halt. But for the data to be useful to businesses, it must be accessible, manageable, and current. At the same time, access to business data must follow certain security practices and oversight. There has been a great amount of work performed in the area of Access Control models, ranging from simple MAC (Mandatory AC) and DAC (Discretionary AC) [11] models to the elaborate RBAC (Role-based AC) [9], and advanced SAC (Semantic AC) [10] and ABAC (Attribute-based AC) [12]. These models have been widely used in business applications in general, and business objects in particular. One of the shortcomings of the access control models is that they are applied to business objects as a whole, and do not provide a fine-grain control over them. For example, a control access model might provide read, write, and delete access on a given business object, but does not provide a vehicle for individual read or write of the business object's individual fields. In this paper we propose a pattern for fine-grain access-controlled business objects.

KEYWORDS

Business object, fine-grain control, permission-based control, role-based control

---

1. INTRODUCTION AND OVERVIEW

A common business object is an object which is used often by web-based applications and services and is a widely acceptable entity in the running of the business. For example, a purchase order (PO), an invoice, and a customer profile could be considered common business objects, as these are parts of day-to-day business activity, but more importantly these have well-established and commonly accepted attributes and behavior. Often a business object is intended for group of people, and should not be accessible for general viewing, creation or updates. However, the precise access policy for a given business object might not be available at design time. Furthermore, the business object access policy itself might change overtime in response to changes in business processes and procedures. It is therefore not always feasible or even possible to address the accessibility of a business object during design time or incorporate it into the application programatically. One possible solution is to associate permissions to business objects. This approach provides multiple benefits for business object designers as well as business applications administrators. At design time, the developers do not need to know the specifics of the permissions, just that a business object they are designing will have permissions associated with it. In addition to business object level, permissions can also be used for field level access and fine-grain control of business objects. For example, a permission can be used to control business object's fields visibility and/or editability within an application. The fine-grain control to business object via permissions can be very beneficial to business applications administrators and business process owners. This paper introduces three patterns for fine-grain access-control to business objects via the use of permissions.

We present the basic principles for fine grained access control for business objects that could be used to design systems where access to objects, their attributes, and operations, must be controlled at an individual level. The base *Pattern for Fine Grain Access Controlled Business Objects* describes the solution at the most general level. At its core, the nature of permissions for the object, subject, attribute, and operation, and the authorization mechanism are left unspecified.

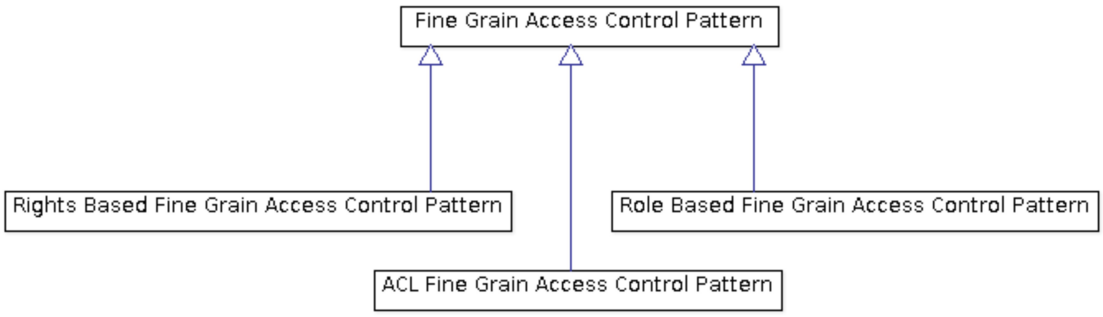

Figure 1: Pattern system for fine grain access control for business objects.

The pattern system from Figure 1includes four patterns and can be expended to cover more cases with access control systems that are more sophisticated. The pattern system includes the specialized *Rights Based Pattern* relies on a simple CRUDC access rights, the Role Based Fine Grain Access Control pattern, and the Access Control List (ACL) Fine Grain Access Control pattern, that implement various permission mechanisms abstracted in the top-level *Pattern for Fine Grain Access Controlled Business Objects.*

2. FINE-GRAIN ACCESS CONTROL PATTERN

2.1 Intent

A holistic access control to business objects is not sufficient as business objects are often comprised of multiple parts (fields) and may contain references to other business objects. Objects need to be designed with built-in support for fine-grained access control to their elements. Consequently, we need a model which provides a fine-grain access-controlled business objects. In this pattern we present an approach for access control to an object and to its individual attributes and operations.

2.2 Example

Consider a simple customer business object, which consists of customer name, address, telephone, email, credit card info, and ordering history.

The customer data should only be available to individuals who need it and have the right (permission) to access it. Most of the time, only parts of the customer's data is needed by those who access it. For example, a marketing department might need to see the customer's order history and contact info, but they should not be able to see the customer's credit card info. On the other hand, the ordering department should have access to the customer's credit card info in order to be able to validate it, but might not have a valid business reason to see the order history.

Access to the customer object and its fields could be programmed into the object itself. This approach however is not desirable as it does not take into account constantly changing business conditions and processes. Any changes to the way customer object is accessed and controlled would require additional programming, which is often expensive and time consuming.

An alternative is to associate permissions with the customer object. Thus, for example, a Marketing Department permission can be associated with the customer business object, and its contact info and ordering history details. At the same time, an Ordering Department permission can be associated with the customer business object's credit card details. Both permissions are associated with the customer object, but each propagates to a different set of its fields, thus each permission provides a fine-grain access control over the customer object.

2.3 Context

You are designing a new business object and need a way to provide fine-grain access control to the object and its fields when your business object is deployed in your business application. The fine-grain access control should be configurable and verifiable. Finally, the fine-grain access to the business object should also provide control over the business object's fields visibility and editability.

2.4 Problem

Determining business object's access control and fine-grain accessibility of its fields at design time or programmatically is not always feasible or possible. How can I design a business object's fine-grained access control which can be parameterized, and thus configurable at run time or at the time of deployment?

2.5 Solution

The UML class diagram for the pattern's design is shown in Figure 2. The business object metamodel is suitable for different runtime environments (such as web client, web backend, web services, Java virtual machine). It include the following concepts:

- *BusinessDataObject* class: for business objects. Such an object is uniquely identified and all objects from the same class have similar states and behavior, as per the object-oriented paradigm. A class state is defined by its attributes (i.e. properties) and an object is capable to execute a set of operations.
- *Attribute* class: an object of this class describes the name and type of a business object's attribute. The runtime value of an attribute is not addressed in this pattern. A BO may have zero or more Attribute objects, one for each of its attribute.
- *Operation* class: each operation supported by a BO is described by an Operation object. Its attributes may include name, return value, parameter list.

Access rights and permissions are represented in this design by corresponding classes for BOs, their attributes, and operations. A BO instance has an optional *DataAccessRightsDescriptor* object that represents the rights assigned to the BO in an abstract and opaque way. An *Attribute* object is also associated with an optional *DataAccessRightsDescriptor* instance, describing the access rights endowed to this particular attribute. The *OperationRightsDescriptor* class describes the rights assigned for a particular operation for a BO, again in an abstract way.

The *Subject* class represents the entity that access a BO's attributes and executes its operations. A *Subject*'s access permissions are described by a *PermissionsDescriptor* object.

The *Runtime* class abstracts the runtime system (e.g. web application server, JVM, browser Javascript engine) and aggregates subjects and BO instances.

The *PermissionAuthority* object is responsible for authorizing access by *Subject*s to BOs, attributes, and operations depending on the access rights descriptors associated for a particular access attempt by a *Subject*. The *PermissionAuthority checkDataAccess*() and *checkOperationAccess*() methods implement the validation for access to a BO's attribute and operation, respectively, and encapsulate policies for dealing with defaults and conflicts.

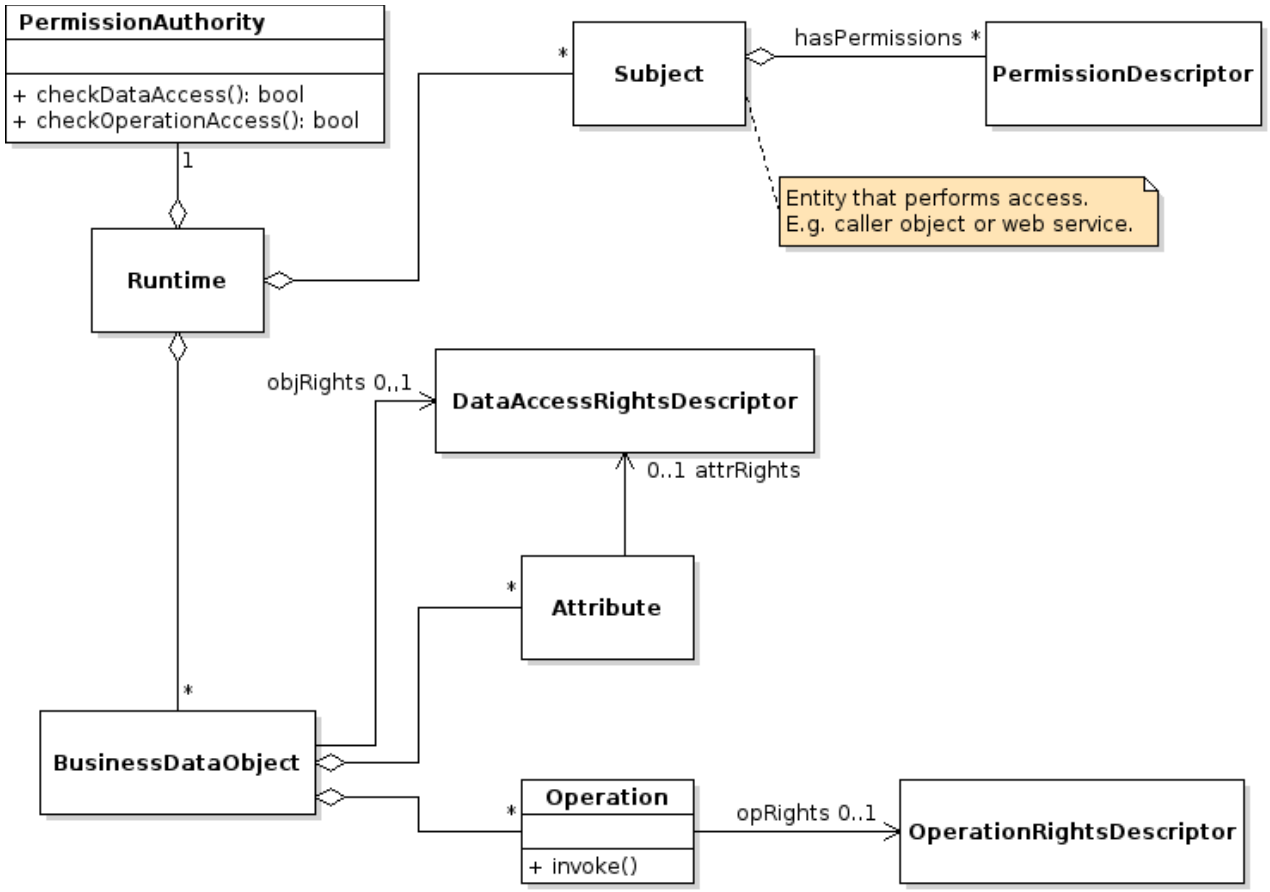

Figure 2: The design of the Fine Grain Access Control for Business Objects pattern.

2.6 Known uses

Most, if not all major Commercial Off The Shelf (COTS) Enterprise Resource Planning (ERP) developers such as Oracle and SAP to name the two biggest, employ some sort of permission-based access to their data forms and data objects. They too apply permission-based access at both the full object level, as well individual object fields. Ariba for example uses permission objects which can be assigned to role objects, which in turn are then assigned to user objects. Developers can then assigned permissions to actions such as editibility and visibility of the fields within a given object. This allows Ariba business objects to be accessed via permissions, without the need for further coding.

3. ROLE BASED FINE GRAIN ACCESS CONTROL PATTERN

3.1 Intent

Presents an approach for access control to a business data object and its individual attributes and operations, that depends on subject's roles and the access rights assigned for specific roles to the object, attributes, and operations.

3.2 Context

Any environment where subjects (users, clients) are assigned to roles and access to protected business objects, their attributes, and operations must be authorized differently based on the subject's roles.

3.3 Problem

Subjects must access protected objects, their attributes, and execute their operations with discrete level of access control and in a way that is scalable and convenient with a large number of objects and subjects. Access permission is granted by a central authority.

3.4 Solution

This pattern, shown in Figure 3, is derived from the more abstract pattern for Fine Grain Access Controlled Business Objects, described above. The access right information for business objects (*DataAccessRight* class) and attributes (*OperationExecuteRight* class) are associated with roles identified by *Role* objects. The *accessType* attribute indicates the rights type (e.g. read, update) that are possible for subjects with a particular role. Similarly, the *OperationExecuteRight* object specifies that subjects with the associated *Role* has the right to execute the corresponding operation.

Roles form a hierarchy defined by the *subRoleOf* association: a subject member of a subrole A is considered to be a member of role B if there exists an association A *subRoleOf* B between objects A and B.

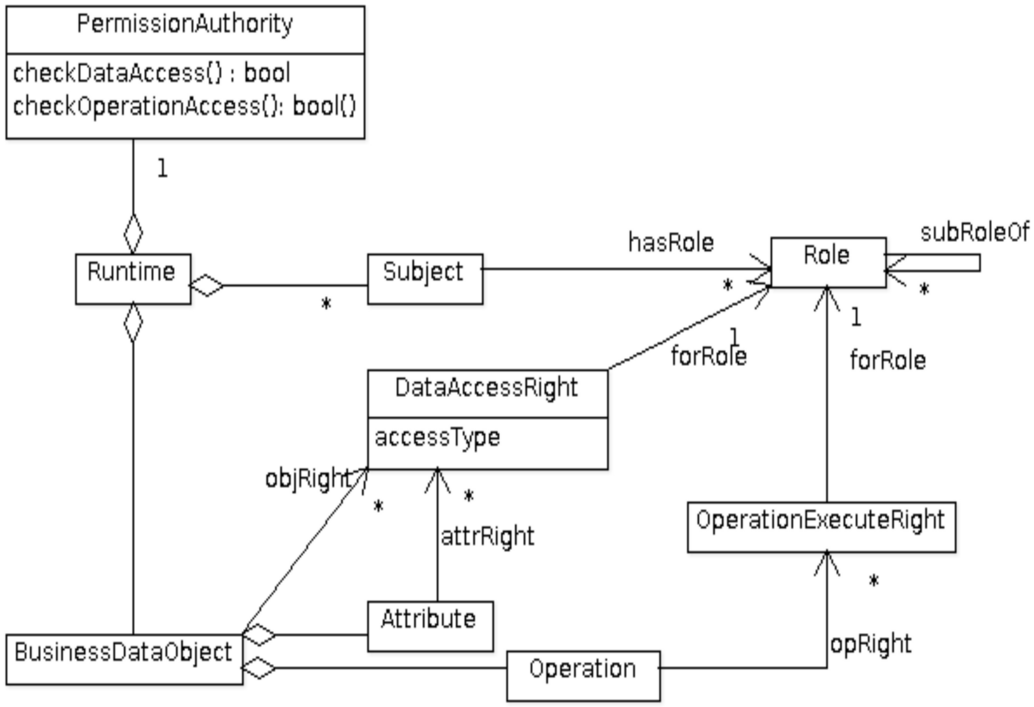

Figure 3: Class diagram for the Role Based Fine Grain Access Control Pattern

With a basic approach, the *PermissionAuthority* object validates access rights to attributes on behalf of the *Runtime* with the following pseudocode algorithm. Note that subject role membership must consider the recursive subrole association.

```
bool checkDataAccess(BusinessDataObject obj, Attribute attr, Subject sub, AccessType type) {
  for (or in obj.objRight) {
    if (or.accessType matches type) and (sub is a member of or.forRole) {
      // subject can access the object. Can it also access the attribute?
      for (ar in attr.attrRight) {
        if (ar.accessType matches type) and (sub is a member of ar.forRole)
          return TRUE   // access granted
      }
    }
```

```
  }
  return FALSE
}
```

A similar algorithm validates subject access to operation execution.

3.5 Consequences

Using roles allows the application developers and administrators to reduce the effort for specifying access permissions as large groups of subjects (e.g. users, web clients) can be assigned to a relatively small number of roles. In addition, access rights for large sets of objects (and their fields) can be handled efficiently through automation and default policies. A per-field rights mechanism provides a much finer level of access control that is suitable for current large web-based systems.

3.6 Known Uses

Unix file access rights is based on a combination of groups, users, owners, and permissions. Users are organized into groups, and each group is assigned read, write and execute permissions.

4. RIGHTS BASED FINE GRAIN ACCESS CONTROL PATTERN

4.1 Intent

A basic approach for access control that requires granular permissions to objects, attributes, and operations in environments with a manageable number of subjects.

4.2 Context

Business objects and their attributes require rights for specific types of access, such as create/read/update/delete and copy. The access right for an operation is to execute it. The environment makes it feasible to assign access rights to subjects for specific objects, attributes, and their operations. This could be an execution runtime (OS), a virtual machine, or a distributed (web) system not open to external clients, for instance.

4.3 Problem

Access to Business Objects, their attributes, and operations must be given individually to subjects in a granular way. A subject may be given access to an object, but not to all its attributes or operations. Also, permissions to read, update, copy, create, and delete must be differentiated – per attribute. The system must apply policies to deal with situations where a subject's rights for an object conflicts with the specific access rights of that subject for the required attribute/operation. For instance, a subject that has the *Read* access right for an object may lack the *Read* right for an attribute it needs to read. One incarnation of this pattern may apply the principle of least privilege and deny access.

4.4 Solution

The design in Figure 4 assigns *RightsDescriptor* objects to a subject that describe specific access type allowed for the subject to individual objects, their attributes, and operations. An optional *RightsDescriptor* object is assigned to each BO, attribute, and operation to describe access rights that apply for *all* subjects that request access. The *PermissionAuthority* object validates access by matching the rights a subject has for a specific object, attribute, or operation with the (optional) rights descriptor owned by corresponding object, attribute, or operation. Validation must consider cases when a subject does not have access rights objects for the requested access and when the object (or attribute/operation) has no rights descriptor. Best practices (assigning least privileges) can be applied.

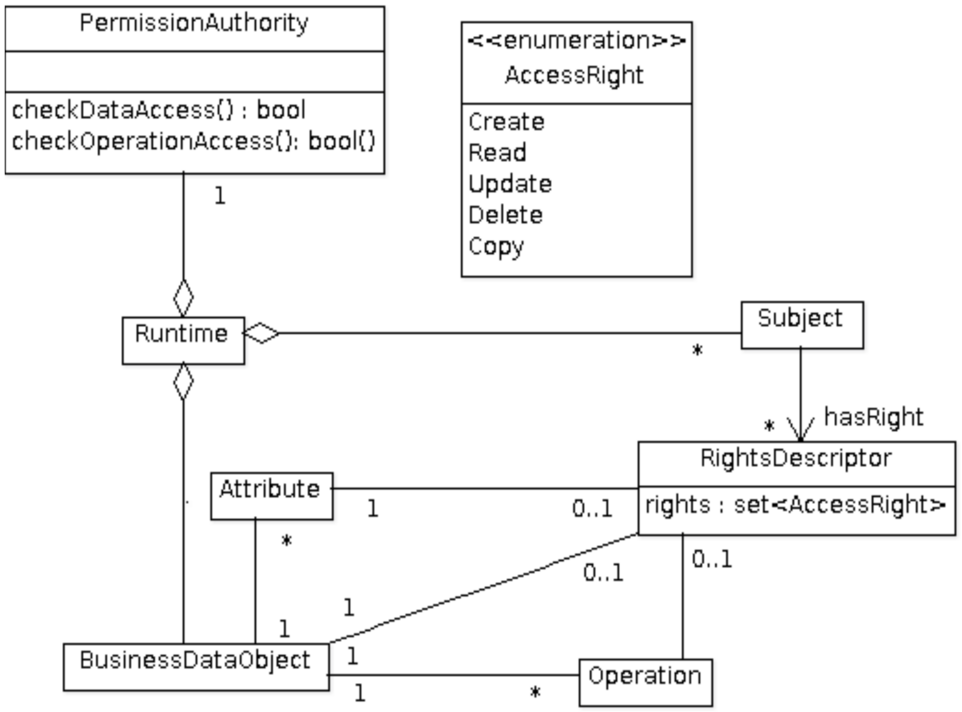

Figure 4: The class diagram for the Rights Based Fine Grain Access Control pattern.

There are 5 access types which control how an instance of a business object can be accessed and what the accessor can do to that instance of a business object. The basis for the access types has been borrowed from the database CRUD operations (Create, Read, Update, Delete). The 5 types applicable to the common business objects instances are CRUDC (Create, Read, Update, Delete, Copy).

For each business object we can map the five access types to permissions which controls the particular access type. This enables us to have a fine-grain access control over any business object. Furthermore, the actual mapping does not have to take place at design time, and can be performed at the time the business object is deployed to the business application.

At design time we create each business object with a set of attributes which serve as place holders for the actual permission mappings when the business object is deployed.

Thus it is expected that each business object will have 5 permission attributes:

Create Permission. A permission value mapped to this attribute indicates that users with this permission are allowed to create the given business object. Creating a new instance of a business object should not be a widely open process available to everyone. Only certain users should be allowed to create new instances of business objects, and this attributes would hold a set of permission values which allow the Create operation.

Read Permission. Another word for reading an object is viewing it. As business objects are comprised of business data, the ability to view their contents is essential. However, there are numerous cases when the business objects should only be viewed for certain group of people, and should not be visible to anyone outside of that group. For that reason the Read Permission attributes allows for mapping the object's visibility (readability) to a permission or a set of permissions. By the same token, there are times when an object should be visible to a wide set of audience, but some of its fields should be hidden from that audience, and only available to a small group of people. A social security number on an employee object or a credit card info on a customer object are examples of such fields. In those cases we can map a Read permission(s) to individual fields, which would override the Read permission(s) at the object level.

Update Permission. Updating business object is a daily occurrence within any organization. For example, an employee's marital status changes and needs to be reflected in the employee object, or a customer address changed, and needs to be updated in the customer object. However, the ability to update business objects should not be granted to everyone. A clerk in the warehouse should be able to view customer address for shipping purposes, but should not be able to update it. The update access control can be solved via permissions. By mapping business object's Update Permission attribute to a permission (or a set of permissions), we can control who can update the given business object. We can also apply are more fine-grain control to business fields by applying a different permission (or a set of permissions) to those fields which should only be accessible for update to a narrower audience (see example in Table 1).

Delete Permission. Deleting a business object is not the same operation as deleting the object in a programming language. Deleting a business object may require deleting some language objects, deleting some database entries, etc. Deleting a business object should be the least used action, and least accessible action, but it is an extremely sensitive one. Deleting business objects is highly discouraged and should only take place in rare circumstances. Some business applications, such as Ariba, do not allow any deletion of business objects. Instead they "deactivate" unwanted business objects within the application, thereby rendering them unavailable to the end users and processes. Even in the case of deactivation (logical delete), only a very narrow set of audience should have ability to delete business objects. Mapping a permission to Delete Permission attributes will accomplish that limited access. Delete operation can also be applied at business object field level. Note: in this pattern delete of field data is different from updating of field data. Updating field involves changing field value from one value to another (i.e. changing employee marital status from Single to Married). Deleting field value is removing any value within that field and leaving it blank (or nill or null).

Copy Permission. Copy is probably the least utilized action in most business applications, yet could greatly increase user productivity and decrease data errors. For example, when reordering office supplies, it would be much quicker to copy an existing order and just modify item quantities as opposed to creating a new order from scratch, especially if the order has a large number of items. Not only does copy offer much faster way to order the needed supplies (user productivity), it also eliminates errors which could have been introduced has the order been created from scratch. At the filed level copy is most often utilized when a business object contains data of Master/Detail type. Copy can be used to copy a detail field (which often contains numerous fields).

All object level permissions are propagated by to the attributes and operations of the object. At the attribute and operation level, these permissions can be overridden by explicitly assigning another permission to an attribute or operation.

At design time we do not need to assign any value to each of the above attributes. Rather, each attribute will be mapped to a permission via parameter.

The same set of attributes is also added to individual fields within the business object at design time. The attributes at the object level and at the object field level allow for a fine-grain control over the business object and its fields at run-time, without the need to know the actual permissions at design time.

The following table provides an example of how permissions can be mapped to a business object and its fields. We use "/" to denote object/field relationship. For example, a customer object and customer credit card would be denoted as Customer/CreditCard.

	Customer Object	Customer/CreditCard Field	Customer/Telephone Field
Create Permission	CustomerService		
Read Permission	CustomerService or Finance	Finance	
Update Permission	CustomerService or Finance	Finance	
Delete Permission	Finance		
Copy Permission	CustomerService		

Table 1: Permission-based field access

Above table shows permission mappings for Customer object and its fields. Users with CustomerService permission are allowed to create, read, update, and copy a customer object, however only users with Finance permission can delete a customer object. Because Customer.Telephone field does not have any permissions assigned explicitly, it inherits the access permissions of the object itself. The field Customer/CreaditCard on the other hand can only be updated or viewed by users with Finance permission. Consequently, users with CustomerService permission may enter customer credit card info at the time of customer business object creation, but do not have access to updated or view customer credit card info subsequently.

Thus via permissions we can provide a fine-grain access control to a business object as a whole, and to its individual fields.

The permissions associated with the given business object access do not have to be determined at design-time. Rather, the mapping of permissions to objects should be parameterized, thereby enabling the creation of the associations at run-time. Assigning permissions at run-time also enables us to group the business objects based on the permission(s) to which they are mapped.

As our intention is to provide fine-grain access control to business objects, we need to define additional permissions for the object attributes and the object operations. These are different from the object permissions, but must be related to them at least by a default propagation scheme. Object attributes will have these permission attributes: *Create Permission.* A permission value mapped to this attribute indicates that users with this permission are allowed to create the given business object attributes. This is only possible for those attributes that allow this by their structure.

Read Permission. Another word for reading an object attribute is viewing it. A permission to read an operation allows access to the history of operation calls. This is a common and useful action in business objects. For instance, consider an account object. It is frequent that there is a need to consult the account transactions.

Update Permission. Updating business object attribute is a daily occurrence within any organization. For example, using the the example above, an employee's marital status changes and needs to be reflected in the employee object, this action needs permission to update the object, as well as permission to update the “MaritalStatus” attribute.

Delete Permission. Deleting a business object attribute is only possible for those attributes that allow this by their structure.

As for operations, we also need specific permissions that are different from the object permissions, but must be related to them at least by a default propagation scheme. Object operations will have these permission attributes:

Call Permission. This is a permission that allows invoking the given operation on the given object instance. Calling an operation requires a permission for updating the object instance, plus a permission to call the operation.

Finally, we should define default propagation rules for permissions so that when an object is assigned permissions the attributes and operations are consistently assigned their corresponding permissions.

Because we can group permissions into roles, we can easily incorporate our configuration into an existing Role Based Access Control (RBAC) type architectures.

4.5 Consequences

This pattern is suitable for systems where it is feasible to assign all subjects rights descriptors for objects they would access, including for their attributes, operations. For each object (and each of its fields) it only requires optional specification for just one rights descriptor that applies to *all* subjects. This mechanism is simpler, but gives less control compared to the role-based alternative, and alsoe does not scale with a large number of subjects.

5. RELATED PATTERNS AND FRAMEWORKS

5.1 Discretionary Access Control (DAC)

The DAC pattern enforces access control based on user identities and the ownership of objects. The owner of an object may grant permission to another user to access the object, and the granted user may further delegate the permission to a third person. [11]

5.2 Mandatory Access Control (MAC)

The MAC pattern governs access based on the security level of subjects (e.g., users) and objects (e.g., data). Ac-cess to an object is granted only if the security levels of the subject and the object satisfy certain constraints. The MAC pattern is also known as multilevel security model and lattice-based access control. [11]

5.3 RoleBased Access Control (RBAC)

The RBAC pattern enforces access control based on roles. A role is given a set of permissions, and the users assigned to the role acquires the permissions given to the role. Since the RBAC pattern is based on roles which are in general fewer than the number of users, it is useful for managing a large number of users. [9]

5.4 Attrubite-based Access Control (ABAC)

Attribute-based access control defines an access control architecture whereby access rights are granted to users through the use of policies which combine attributes together. The policies can then use any type of attributes (user attributes, resource attribute, etc...). Attributes can be compared to static values or to one another thus enabling relation-based access control. [12]

5.5 Semantic Access Control (SAC)

The Semantic Access control model was created in 2002. The fundamentals of this semantics-based access control model are the definition of several metadata models at different layers of the Semantic Web. Each component of SAC represents the semantic model of a component of the access control system. The semantic properties contained in the different metadata models are used for the specification of access control criteria, dynamic policy allocation, parameter instantiation and policy validation processes. [13]

6. CONCLUSIONS AND FUTURE WORK

The permission-based access control to common business objects and their entities provides a clear, generic and extendible design-time approach to securing access to business data. It allows for use of as many or as fine-grained permissions to control access to various parts of business data in a declarative way.

It is important to note that the permissions are independent of the access control model used. The content of the permission-specific attributes can be adapted to each access control model. For instance, in a Mandatory AC model the content of a permission attribute would be the security level, in the Discretionary AC model it would point to specific users, in RBAC it would contain roles, and finally in the Semantic AC model, it would point to AC policies.

7. ACKNOWLEDGMENTS

We would like to thank Antonio Maña for his invaluable feedback and support during the shepherding of this paper.

8. REFERENCES

[1] A Guide To The Sarbanes-Oxley Act, The Sarbanes-Oxley Act of 2002, DOI=http://www.soxlaw.com/

[2] Fowler M., 1997. Analysis Patterns: Reusable Object Models, Addison-Wesley Longman.

[3] Daum B., 2003. Modeling Business Objecs with XML Schema, Morgan Kaufmann Publishers.

[4] Eriksson H.-E., Penker M., 2000, Business Modeling with UML: Business Patterns at Work, OMG Press / Wiley Computer Publishing

[5] Adams J., Koushik S., Vasudeva G., Calambos G., 2002, Patterns for e-business: A Strategy for Reuse, IBM Press

[6] Buckl S., Matthes F., Monahov I., Roth S., Schulz C., Schweda C.M., 2012. Enterprise Architecture Management Patterns for Company-wide Access Views on Business Objects, ACM Transactions on Applied Perception, Vol. 2, No. 3, Article 1, Publication date: May 2012.

[7] Chang K.Y., Chen L.S., Lai C.K., 1999. Document-View-Presentation Pattern, Department of Electrical Engineering, National Cheng-Kung University, Taiwan.

[8] E. B. Fernandez and Pan R. A Pattern Language for Security Models. In Proceedings of the 8th Conference on Pattern Language of Programs (PloP), Monticello, IL, 2001.

[9] D. Ferraiolo, D. R. Kuhn, and R. Chandramouli. Role-Based Access Control. Artech House, 2003.

[10] Yagüe, M. I., & Maña, A. (2005). Semantic access control model: A formal specification. In Computer Security–ESORICS 2005 (pp. 24-43). Springer Berlin Heidelberg.

[11] Kim, D. K., Mehta, P., & Gokhale, P. (2006, October). Describing access control models as design patterns using roles. In Proceedings of the 2006 conference on Pattern languages of programs (p. 11). ACM.

[12] Yuan, E., & Tong, J. (2005, July). Attributed based access control (ABAC) for web services. In Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on. IEEE.

[13] Semantic Access Control, A Semantics-based Access Control Model for Open and Distributed Environments, http://www.lcc.uma.es/~yague/Semantics-basedAccessControl.html