View Connectors for the integration of Domain Specific Access Control

Tine Verhanneman*, Frank Piessens, Bart De Win and Wouter Joosen
DistriNet, K.U.Leuven,
3001 Leuven, Belgium
{tine,frank,bartd,wouter}@cs.kuleuven.ac.be

Abstract

Engineering access control in distributed applications is a challenging problem for many application domains. It should be possible to set and manage one organization-wide access control policy that must then be enforced reliably in a multitude of applications running within the organization. This is severely complicated by the fact that an access control policy can be fine-grained and dependent on application state, and hence its enforcement can crosscut an application in an intricate way.

Based on the observation that the access control enforcement points in an application are essentially pointcuts, this paper proposes a new approach for engineering access control into applications.

Experience with the access control concern also clearly shows the need for full life-cycle support for aspects.

1 Introduction

Engineering access control in distributed applications is a challenging problem for many application domains [3]. It should be possible to set and manage one organization-wide access control policy that must then be enforced reliably in a multitude of applications running within the organization. This is severely complicated by the fact that an access control policy can be fine-grained and dependent on application state, and hence its enforcement can crosscut an application in an intricate way.

State-of-the-art commercial systems, such as Tivoli Access Manager [12], provide for centralized management of access control, but as soon as policies are dependent on application state, explicit calls to the authorization engine must be inserted in the application code.

Based on the observation that the access control enforcement points in an application are essentially pointcuts [9], this paper proposes a new approach for engineering access control into applications. Our approach is based on the concepts of access interface and view connector, presented in figure 1:

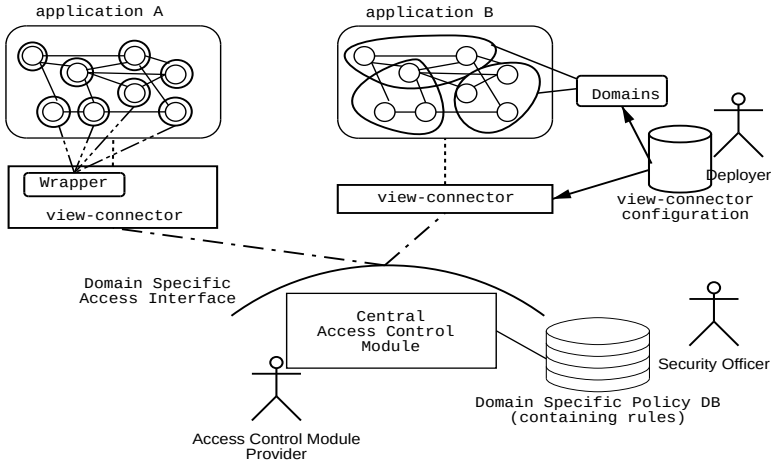
The *Access Interface* represents an abstract view on an application reflecting only access control relevant information. Such an access interface should be relatively constant within one organization, but could be application domainspecific (for instance medical organizations will probably use a different access interface than financial organizations). Through this interface, a centrally managed and configured access control module receives notifications of access attempts from applications, and can query applications for application state while deciding whether these accesses should be allowed or not. The access control module can be configured by means of declarative policy rules that specify the access control policy in terms of the access interface.

View connectors bind the access interface to different applications. They map application specific concepts to the domain concepts represented in the Access Interface and assign object instances to domains, which constitutes a group of objects, governed by the same set of access rules. By implementing view connectors using a dynamic, wrapper based AOSD system, applications can be connected to an access control module at deployment time. The view connector can be made configurable too, so that the mapping of application concepts to domain concepts can be set declaratively.

The rest of the paper is organized as follows: the next section elaborates on the access interface concept and the way access rules are set up in terms of this interface. How this abstract view is tied to a particular application by means of a view connector is shown in section 3. In section 4, a possible implementation is described. We discuss future work and related work in sections 5 and 6. and draw conclusions in section 7

*Tine Verhanneman is supported by a grant from the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen)

Figure 1: Overview



2 Access Control View

In this section, a simplified application is presented to illustrate the introduced notion of an Access Interface. Thereafter, the key concepts of this interface are set forth and illustrated by means of a rudimentary policy.

Figure 2 depicts the class diagram of a simplified medical application, which is inspired by the HL7 Reference Information Model [10]. Herein a `Person` can adopt several `Role`s, such as for example the `Patient` and/or `Employee` role. `Act` represent medical examinations, observations, ... To support, for instance, causal relations and reasoning, `Act`s can be linked by means of an `ActRelationship`

Based on this model we can now formulate access control rules, which can be made quite complex, as access decision may depend on object state, subject state, environment, the invoked operation (with its parameters) and additional access control attributes, subject and objects have been labeled with. The following discussion addresses policy rules including only object state and access control attributes, like the access rule described below:

> Only members of the medical staff can view and modify medical data. Patients, on the other hand, can only view their own medical data.
>
> Medical data are contained in the classes `Act`, `ActRelationship` and the attribute `Person.disabilityCode`.

Access Interface An access interface provides a domain specific and abstract view on the application, only displaying and adding information relevant for access control. For example, concepts like "high risk transactions" for financial applications; or "emergency access" for healthcare applications can be defined in an access interface. The information exposed by this abstract view may support both generalizing rules affecting a large group of objects as fine-grained rules pinpointing specific objects. So, the granularity of the access control is determined by the access interface, just like the terms the access rules are formulated with.

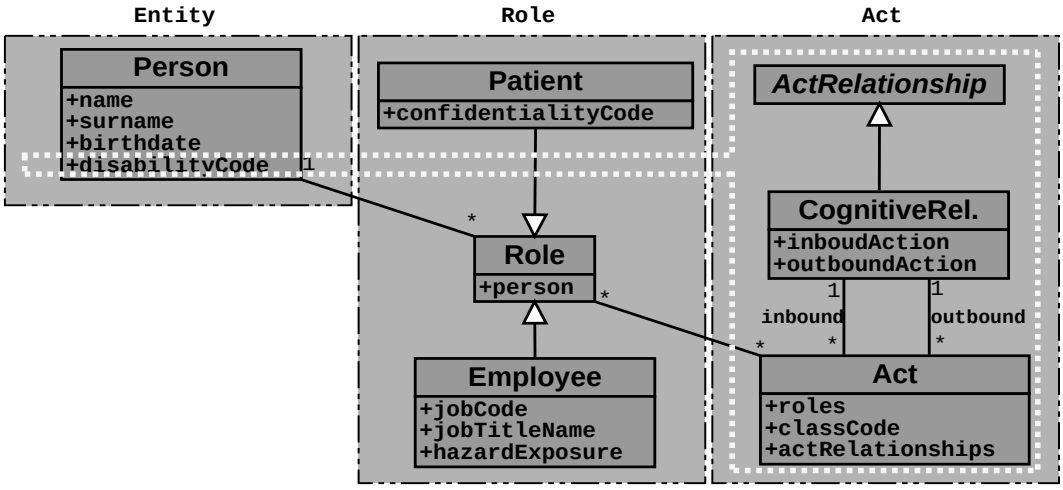
At present, two sorts of terms have been defined: attributes and actions.

- *Attributes:* In fact, we can distinguish *access control attributes* and *application attributes*. The former augment the object's state with additional security related information, such as roles and data sensitivity level. The latter represent state information that is not security-related by nature, but can influence the access control decision; e.g. the vip-status of a person may command the enforcement of another access control policy. The attributes must be produced by the objects implementing the "Access Interface" whenever the access control requests them. What is more, access control rules may only be set up in terms of these attributes.
- *Actions:* The actions in the Access Interface enumerate the permissions that are relevant for the access control policy rules. Modifiers, accessors and constructors are some straigthforward examples of actions.

What has been omitted from this interface are conditions, e.g time-constraints, or provisional constraints. It should be fairly easy to include these.

```
<access-interface>
   <interface>MedicalData</interface>
   <attributes>
      <attribute>owner</attribute>
```

Figure 2: Classdiagram




```
   </attributes>
   <actions>
      <action>view</action>
      ...
   </actions>
</access-interface>
```

Policy Rules We restrict ourselves only to positive authorization policy rules, defining the actions that subjects are permitted to perform. Below, the example policyrule is written down in terms of the Access Interface. Some clarification is needed for the domain-element, which constitutes the collection of objects the rule is to be applied to. How objects are assigned to domains is discussed in further details in section 3.

```
<policy-rule>
   <domain>PatientData</domain>
   <access-interface>
      MedicalData
   </access-interface>
   <subject>MedicalData.owner</subject>
   <actionlist>
      <action>view</action>
      ...
   </actionlist>
</policy-rule>
```

The constructs, elaborated on in this section, support a high-level view on access control, enabling a person who has no extensive expertise into the implementation of the application, to draw up the access control policy. In the next section, the concrete mapping of the `Access Interface` and `Policy Rule` onto the application is discussed in further detail.

3 View Connector

A view connector essentially binds an access interface to a particular application. Two concepts can be identified in relation to this connector:

View-Connector A view-connector indicates for an object type how the values of the attributes in the access interface should be retrieved, and also how the methodinvocations in the object's interface can be mapped onto the actions in the access interface.

```
<view-connector>
   <type>Act</type>
   <access-interface>
      MedicalData
   </access-interface>
   <attributes>
      <attribute>
         <name>owner</name>
         <value>Act.getPatient.getPerson</value>
      </attribute>
   </attributes>
   <actionlist>
      <action>
         <name>view</name>
         <value>get*</value>
      </action>
      ....
   </actionlist>
</view-connector>
```

Domain-Membership A domain represents a set of target objects of an access rule. This set can in principle contain objects of varying type. What is more, domain-membership can be determined based on (dynamic) parameters, like for example deployment location. In the following example, the domain

PatientData is being defined as all objects of the type `Act`, `CognitiveActRelationship` and also as the value of the attribute `Person.disabilityCode`.

```
<domain>
   <name>PatientData</name>
   <list>
        <type-attribute>
           Person.disabilityCode
        </type-attribute>
        <type>Act</type>
        <type>ActRelationship</type>
   </list>
</domain>
```

For every policy rule, it has to be verified that there exists a view-connector for every type of which instances belong to the rule's domain. As long as domain-membership is determined by static criteria, this verification should be fairly simple.

Secondly, in order to be able to pinpoint specific objects and assign them to a domain, support for naming object-instances should be provided.

4 Prototype

In this section, a possible implementation of the access control logic is suggested. First, the overall architecture is described. Subsequently, we describe into further detail how a dynamic wrapper based AOSD system, such as for instance JAC [15] and Lasagne [16], can be used for the implementation of view connectors to connect the access control module at deployment time to the application. The access module itself, will not be dealt with in great depth, because this can be designed with existing object oriented/framework methodologies.

Architecture Figure 3 depicts an overall overview of the architecture of the access control elements. Access control policies are administered centrally in a Policy Database by the Management server. Whenever a target is accessed, the invocation is intercepted by the Reference Monitor, which passes on the reified method invocation, along with additional information, such as the client's identity, to the Access Control Decision Function. Possibly the latter requires additional information from the target, which is retrieved by means of the Attribute Retriever.

Generic Implementation The presented approach is generic, in the sense that the wrappers are only used to intercept the method invocation, and to redirect the reified method invocation to the access control module, i.e. merely as reference monitor. This approach allows for a generic wrapper, which is nor type nor instance specific, but on the other hand stresses the access control module extensively.

Figure 4 presents the collaboration diagram of the access logic:

1. *Rule Matching:* In a first step, all the rules must be selected which are to be applied to this method invocation. The domain(s) the target belongs to can be retrieved by means of its identity and the membership mapping, discussed in section 3. The view connector conveys which action the method invocation maps to. By means of the domain and the action, we can retrieve the rules, introduced in section 2 that are applicable. For reasons of convenience, we will call this module the "RuleMatcher".
2. *Attribute Retrieval:* Once the applicable rules have been retrieved, the necessary attributes, needed by the access rule, must be filled out by means of the attribute mapping, introduced in the view connector.
3. *Access Control Decision:* Finally, if all information is available to the Access Control Decision Function, the latter can allow or deny access.

5 Future Work

In the previous sections, a powerful approach is presented, to disentangle access logic from application logic, while supporting at the same time complex access rules based on application state. However, extensions and improvements can still be made. Below, an overview of further work is given:

Access Decision Information In section 2 the only application state we have taken into account are object state and object access control attributes. Our objective is to extend the information, the access decision function has at its disposal. Below, one of these extensions are discussed:

Parameters of methods: In the Access Interface, the notion of an action has been included, but the parameters of these methods have been neglected until now. Sometimes it may, however, be necessary to take into account the value of these parameters, while selecting the access rules to enforce. For example, the amount of money to be transferred in a transaction, may determine which policy rule is applicable. One possibility, is to extend the notion of an action in the access interface so that it also includes parameters. Another possibility, is to introduce the concepts "LowRiskTransaction" and "HighRiskTransaction" in the access interface and to fill out the treshhold values in the view connector. In fact, both options may be valuable.

Domain Membership In section 3, it has already been pointed out that it has to be known at

Figure 3: Architecture

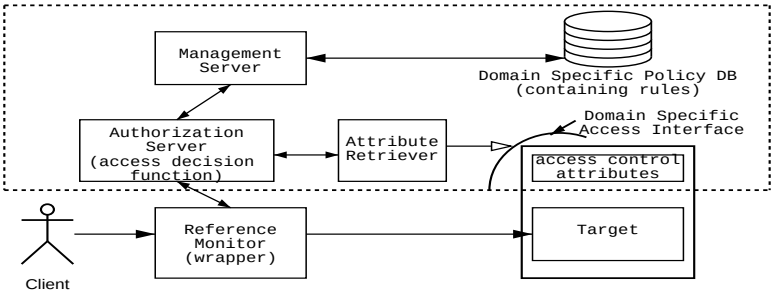
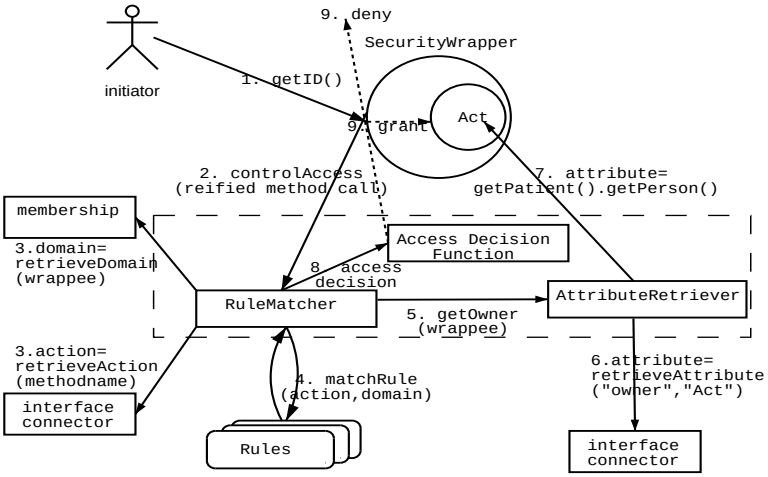


Figure 4: Generic Implementation



deployment time which types are represented within a particular domain to assure that every required view connector has been provided.

Aspect Interference Typically, the access control aspect must be capable of retrieving application state information, but does not need to be able to modify application state. The access control module is a trusted component. So, whenever this module retrieves information from the application to make an access control decision, this access should not be mediated by the reference monitor, i.e. it should be possible to switch the access wrappers "on and off". The same remark holds for other aspects too; it should be verified that accesses initiated by the access module itself, do not cause any undesired side-effects.

Validation Last but not least, it is our objective to validate the presented approach by verifying to what extent the different access models (DAC, MAC and RBAC) can be implemented by means of access interfaces and view connectors.

6 Related Work

The presented work is related with several research domains,which are discussed below.

In the domain of *access control engines* extensive research has been carried out, e.g. the Flexible Authorization Framework (FAF) [11], Woo and Lam [18], and Bertino et al. [1].

The *decoupling* of authorization logic from application logic has been the prime objective of architectures like Resource Access Decision Service (RAD) [4] and the IBM Tivoli Access Manager [12]

With respect to the *binding* of access control functionality to applications, mainstream commercial systems, such as for example J2EE [5] and .NET [13] use fairly simple method-level interception techniques. The usefulness of AOSD techniques for binding access control logic has been shown earlier by B. De Win [7]. He uses the AspectJ tool to modularize this binding. Our view connectors can be seen as a special case of Collaboration Interfaces, proposed by K. Ostermann and M. Mezini [14], tuned to the access control scenario.

7 Conclusion

The definition of an access control view on an application, provides abstractions that can be reused throughout applications based on the same domain model, so that also maintainability and manageability are improved. An access interface only displays information relevant for access control, and therefore facilitates the translation of (high level) requirements into access rules. In a next phase, this access interface can be bound in a natural way to a particular application by means of view connectors. In terms of the access control aspect development and lifecycle, the access control view offers thus an extra step in the reconciliation of the fulfilment of access control requirements with the integration of access control logic in the application and provides a clear separation between the different roles, the security officer, deployer and security module provider play in the overall picture.

Status At the time of writing, no implemented prototype is available yet. Two masterthesis students are, however, working on a realisation in JAC.

References

[1] E. Bertino, F. Buccafurri, E. Ferrari and P. Rullo, *An authorizations model and its formal semantics*, In Proceedings of the 4th European Symposium on Research in Computer Security (ESORICS'98), Louvain-La-Neuve, Belgium

[2] K. Beznosov, *Object Security Attributes: Enabling Application-specific Access Control in Middleware*, 4th International Symposium on Distributed Objects & Applications (DOA), pp. 693-710, Irvine, California, October 28 - November 1, 2002.

[3] K. Beznosov, *Engineering Access Control for Distributed Enterprise Applications*, PhD thesis, Florida International University, 2000

[4] K. Beznosov, Yi Deng, Carol Burt, and John Barkley, *A Resource Access Decision Service for CORBA-based Distributed Systems*, In 15th Annual Computer Security Applications Conference (ACSAC), pages 310-319, Phoenix, Arizona, December 1999. IEEE Computer Society.

[5] S. Bodoff, D. Green, K. Haase, E. Jendrock, M. Pawlan and B. Stearns, *The J2EE Tutorial*

[6] N. Damianou, N. Dulay, E. Lupu, M. Sloman *The Ponder Policy Specification Language*, Proceedings of the International Workshop on Policies for Distributed Systems and Networks, volume 1995 of LNCS, pages 18-38. Springer-Verlag, January 2001

[7] B. De Win, *Engineering Application-Level Security through Aspect-Oriented Software Development*, PhD thesis, Katholieke Universiteit Leuven, 2003

[8] B. De Win, F. Piessens, W. Joosen, and T. Verhanneman, *On the importance of*

the separation-of-concerns principle in secure software engineering, Workshop on the Application of Engineering Principles to System Security Design, WAEPSSD, Boston, MA, USA, November 6-8, 2002, Applied Computer Security Associates (ACSA)

[9] B. De Win, W. Joosen and F. Piessens *Developing Secure Applications through Aspect-Oriented Programming* In: "Aspect-Oriented Software Development", Mehmet Aksit, Siobhan Clarke, Tzilla Elrad, and Robert E. Filman (Eds.), Addison-Wesley, to appear.

[10] Health Level Seven, *http://www.hl7.org*

[11] S. Jajodia, P. Samarati, M.L. Sapino, V.S. Subrahmanian, *Flexible Support for Multiple Access Control Policies*, ACM Transactions on Database Systems, Vol. 26, No.2, June 2001, Pages 214-260

[12] G. Karjoth, *Access Control with IBM Tivoli Access Manager*, ACM Transactions on Information and System Security Vol. 6, No. 2, 232-257 (2003)

[13] J. Lowy, *Decouple Components by Injecting Custom Services into your Object's Interception Chain*, MSDN Magazine, March 2003

[14] M. Mezini and K. Ostermann, *Integrating Independent Components with On-Demand Remodularization*, OOPSLA'02, November 4-8, 2002, Seattle

[15] R. Pawlak, L. Duchien, G. Florin, L. Seinturier *JAC : a Flexible Solution for Aspect Oriented Programming in Java*, Reflection 2001, Kyoto, Japan, September 2001

[16] E. Truyen, B. Vanhaute, W. Joosen, P. Verbaeten, B. N. Joergensen, *Dynamic and Selective Combination of Extensions in Component-based Applications*, In 23rd International Conference on Software Engineering, pages 233-242, Toronto, Ontario, Canada, May 2001, IEEE Computer Society

[17] T. Verhanneman, L. Jaco, B. De Win, F. Piessens, and W. Joosen, *Adaptable Access Control Policies for Medical Information Systems, Distributed Applications and Interoperable Systems*, In J.-B. Stefani, I. Demeure, and D. Hagimont, editors, Distributed Applications and Interoperable Systems, volume 2893 of LNCS, pages 133-140. Springer-Verlag, November 2003

[18] T.Y.C Woo and S.S. Lam, *Authorizations in distributed systems: A new approach*, Journal of Computer Security 2(2-3): 107-136, 1993