Model Driven Security: Unification of Authorization Models for Fine-Grain Access Control [*]

Carol C. Burt
Barrett R. Bryant
University of Alabama Birmingham
cburt, bryant@cis.uab.edu

Rajeev R. Raje
Andrew Olson
Indiana University Purdue University Indianapolis
rraje,aolson@cs.iupui.edu

Mikhail Auguston
Naval Post Graduate School
auguston@cs.nps.navy.mil

Abstract

The research vision of the Unified Component Meta Model Framework (UniFrame) is to develop an infrastructure for components that enables a plug and play component environment where the security contracts are a part of the component description and the security aware middleware is generated by the component integration toolkits. That is, the components providers will define security contracts in addition to the functional contracts. These security contracts will be used to analyze the ability of a service to meet the security constraints when used in a composition of components. A difficulty in progressing the security related aspects of this infrastructure is the lack of a unified access control model that can be leveraged to identify protected resources and access control points at the model level. Existing component technologies utilize various mechanisms for specifying security constraints. This paper will explore issues related to expressing access control requirements of components and the resources they manage. It proposes a platform independent model (PIM) for the access control that can be leveraged to parameterize domain models. It also outlines the analysis necessary to progress a standard transformation from this PIM to three existing Platform Specific Models (PSMs).

1. Introduction

Enterprises are increasingly dependent upon multiple middleware technologies that enable new business paradigms by weaving together legacy systems with advanced technology. Component-based system integration supports core business functionality, integrates business processes and enables companies to communicate with customers, suppliers, and business partners. The Unified Component Meta-Model Framework (UniFrame) [1] attempts to unify distributed component models under a common meta-model for the purpose of enabling the discovery, interoperability, and collaboration of components via generative software techniques. This research targets the dynamic assembly of distributed software systems from components developed using different component models, and explores how the quality of service (QoS) requirements, such as security, influence the design of components and their compositions. The inherent complexity of such integrations introduces significant challenges for controlling access to application resources (business, customer and personal information as well as product and application features). This paper expands on our previous work [2, 3] to explore how Model Driven Architecture techniques may be used for an integration of the access control solutions in heterogeneous environments.

OMG's Model Driven Architecture (MDA) [4] initiative facilitates the standardization of Platform Independent Models (PIMs) and the transformation of those models to multiple Platform Specific Models for implementation. One of the challenges of Model Driven Architecture is the existence of Platform Specific Models that do not adhere to a unified Platform Independent Model. In such cases, bridging is, at best, hand crafted and at worse, impossible. Today this is the case for the access control models. What is needed is a Platform Independent Model for access control that forms the foundation of end-to-end access controls in heterogeneous computing environments. This model must accommodate existing Platform Specific Models while providing the flexibility for innovation in access control technology.

This paper proposes a Platform Independent Model for access control (AC-PIM) that provides a clear architectural separation between the access policy (the management and expression of access rules), the access

[*] This research was supported by the U. S. Office of Naval Research under the award number N00014-01-1-0746.

decision (evaluating policy at a given point in time), and the access control (the enforcement of access decisions). The paper also explores access control models adopted via different standards organizations and outlines the transformations to their access control Platform Specific Models.

2. Relevant Research and Standards

The ITU-T recommendation X.812 (ISO/IEC 10181-3) [5] provides a reference model for the access control that is consistent with the model proposed in this paper. There are also several consortium and de facto standards that are relevant for this work. They are outlined below. The detailed work will select three of these models for analysis.

2.1 Globus GRID Research

Globus [6] is a research project that focuses not only on the issues associated with the building of computational grid infrastructures, but also on the problems that arise in designing and developing applications that use grid services. Globus is developing basic security algorithms for secure group communications, management of trust relationships, and developing new mechanisms for fine-grained access control. Globus authorization requirements and the issues that arise with current authorization technologies in GRID are outlined in [7].

2.2 OASIS

OASIS [8] is a not-for-profit global consortium that drives the development, convergence and adoption of e-business standards. There are two OASIS specifications that are of interest. They are the eXtensible Access Control Markup Language (XACML) [9] and the Security Assertion Markup Language (SAML) [10]. XACML is an XML specification for expressing policies for information access over the Internet. SAML is an XML-based security standard for exchanging the authentication and authorization information.

2.3 Object Management Group (OMG)

The Object Management Group (OMG) [11] is an open membership, not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications. The OMG Resource Access Decision Facility (RAD) [12] provides a uniform way for application systems to implement a fine-grain access control where the protected resources may be physical, logical, or conceptual or understood only within the context of the business application.

2.3 Java and the Java Community Process

Java provides a Java™ Authentication and Authorization Service (JAAS) [13] package that enables services to authenticate and enforce access controls upon users. J2EE and Java Connectors are required to utilize this model. The Java Community Process (JCP) [14] is an open organization of international Java developers and licensees whose charter is to develop and revise Java technology specifications, reference implementations, and technology compatibility kits. Java Specification Request 115 (JSR-115) [15] is progressing a Java Authorization Contract for Containers (JACC). The Java Authorization Contract for Containers (JACC) seeks to define a contract between containers and authorization service providers that will result in the implementation of providers for use by containers.

2.4 Microsoft ASP.NET

ASP.NET [16] supports the traditional methods of performing the access control (file based) and also provides an URL authorization, which allows administrators to provide an XML configuration that allows or denies an access to URLs based on the current user or the role [17]. Developers can explicitly code authorization checks in their application or can take advantage of the common language runtime's support for declarative security. ASP.NET offers an extensible security architecture that allows the developer to write a custom authentication or authorization server.

2.5 NIST Role-Based Access Control

The National Institute of Standards has proposed a voluntary consensus standard for the Role-Based Access Control [18]. The role based access control (RBAC) is a technology that is attracting an increasing attention, particularly for the commercial applications, because of its potential for reducing the complexity and cost of security administration in large networked applications. Since the publication of the Ferraiolo-Kuhn model [19] for RBAC in 1992, most information technology vendors have incorporated RBAC into their product line, and the technology is finding applications in areas ranging from health care to defense, in addition to the mainstream commerce systems for which it was designed. The RBAC has become the de facto standard for access control in component and web environments. This work is the result of the significant NIST research and patents that they hold on the access control technologies [20]. It is our goal for the access control platform independent model to accommodate (but not require) the NIST RBAC model.

3. Access Control Unification Issues

The first step toward Model Driven Architectures that include access control parameterization and/or authorization contracts is the establishment of a common vocabulary. Although generalized frameworks for the access control have established a common vocabulary for operating system enforced access control models [5, 21], there is no standard vocabulary (or several depending on the perspective!) for discussing the modeling elements of the access control across heterogeneous distributed computing and component-based platforms. This is immediately evident after examining the "standards" that have been progressed to enable a fine-grain access control in these platforms. For example, the OMG Resource Access Decision Facility has an "AccessDecision" object (ADO) that provides the "access decisions" based on the "security attributes of a principal", a "named resource" and an "operation" on the resource [22]. The OASIS SAML/XACML access control model defines an "AuthorizationAuthority" which provides the "authorization decisions" based on "attributes of a subject" and an "action" [9, 10]. The Java2 Enterprise Edition (J2EE) model defines a SecurityManager which enforces the access controls and consults with an AccessController that provides the access decisions based on the permissions granted to a Principal (in native Java this is the same model except the permissions are granted to a Codebase) [13].

Fortunately these models contain many architecturally consistent elements; for example, an AccessDecision object, an AuthorizationAuthority, and an AccessController represent the same architectural element in access control architecture. They do not share a common reference model, however, so it is difficult to determine without a significant analysis if they provide equivalent semantics or not. The interface definition languages are also different. The OMG standardizes the data formats and interfaces for requesting access decisions via ISO IDL [23]; the OASIS specification uses XML schemas [24] and the J2EE model uses native Java [13] defined interfaces and data structures. As a result, a business architect and/or developer must be familiar with a variety of access control technologies and platform languages in order to define the end-to-end access control in a heterogeneous environment.

In addition to the diverse vocabulary and specification languages utilized by the existing access control implementations, every layer of technology has an access control model. There are operating system models, database and messaging infrastructure models, and component technology models. The role-based access control (RBAC) has become popular as the access control model for component platforms. RBAC has enjoyed success because it is much more flexible and scalable than the user or group based models and is implemented in most of the available component platforms. It is not, however, sufficient to support many complex business scenarios. For example, the access control policies may require an assessment of additional environmental factors such as time, location, relationships or credit limits which may supplement an RBAC policy. For this reason, RBAC does not provide a complete unification model, rather a specific instance of a model.

There is also an issue related to expressing the access control rules. Unfortunately even when the access control is considered during analysis and design, the requirements are typically expressed in a natural language as business rules. That is, the focus is often to identify the access policy, not to architect the system so that it can accommodate dynamic policy changes. If (when) access policy changes, it becomes a part of the application project to modify the software to update the rules. This adds to the complexity of the access control architecture and makes it impossible to change the access policy without software changes. There are products which support model driven techniques, however, the access control mechanisms are typically expressed only in the platform specific manner (via application code, servlet filters, IIOP interceptors, J2EE deployment descriptors or product specific mechanisms such as graphical interfaces and proprietary API's). Thus, there is no standard way to define the access decision points and/or policy in a platform independent model such that it could be applied consistently across multiple technologies.

The end result is that the task of protecting business resources is increasingly being pushed to the business application developer. Of course, each level of access control still exists and must be administered. A single "application" typically has many "userids" (perhaps the same, perhaps different)" that are utilized in providing the access control across the application. As an example, the JAVA Blueprints Pet Store [25] has multiple userids that must be defined at different infrastructure levels before the application will execute successfully. This sample explores the "best practices" in a system integration architecture utilizing Web Services, Java Components (EJBs), Messaging (JAX/RPC, IIOP, and JMS) and Connectors (JDBC). In addition to multiple userids defined in deployment descriptors, the Pet Store application also supports self-registration of userids that are application specific and completely unknown to J2EE, the web server or the operating system. This "best practices" blueprint architecture documentation suggests that e-business applications must manage userids and the access control [26]. This is an example of the trend of pushing the access control into the application layer. That is, application architects and developers are being forced to include user management, access policies, and programmatic access control logic within their business software. This forces the expenditure of precious business developer resources on building application specific access control infrastructure for managing user information and access control policies, thereby, increasing the cost and time required to create the application.

A component infrastructure that requires exposing knowledge of the underlying access control model to the business developer (via programmatic API's such as isCallerInRole or isUserInRole) has made it difficult to hide the diversity inherent in the access control models when more complex access control policies are required. Although vendor products may extend the RBAC model and/or implement proprietary mechanisms to support more sophisticated access policy, in the absence of an AC-PIM as a reference model for access control, the task of understanding and comparing product features becomes difficult. The task of creating and maintaining consistent policies is also very difficult while a consolidated auditing is near impossible.

Access policies may often change and/or be governed by the legislation that differs from location to location. Business developers should not be required to understand those policies but unfortunately this is what happens during today's application development process. These issues limit the ability of a developer to use components in dynamic system compositions where the access control policy may be significantly different from what was provided in the original usage of that component. We will explore how the proposed model shifts the majority of this work to the provider of the infrastructure authorization service software and discuss how future component infrastructure could assist in assuming more of this responsibility.

4. Goals for Model Driven Access Control

To fully realize the potential of Model Driven Architecture for the access control, an access control platform independent model and the mechanism for the parameterization of domain platform independent models with access control points must be standardized within the OMG Model Driven Architecture roadmap. The goal of the access control platform independent model (AC-PIM) is to provide an abstract view of the access control that can be utilized at the modeling level for the parameterization of domain models. This will enable transformations to access control platform specific models (AC-PSM) that incorporate access control points.

The paper begins the analysis necessary for the unification of the access control models by identifying the vocabulary and abstractions that can be standardized for the purpose of model parameterization (thus enabling a transformation to existing access control models) and the common feature support to ease the secure interoperability. Thus, the goal of the proposed research is the creation of a unifying AC-PIM from which existing security models can be mapped and/or bridged. This will simplify the task of the middleware when cooperation is required to meet the underlying security constraints (such as the delegation of credentials and/or requesting access control decisions based on a local policy). For the UniFrame project, a goal is to identify the work necessary for enabling the generation of access control bridges for heterogeneous system compositions. That is, we wish to provide the foundation for new research projects in using the generative techniques for access control and secure interoperability.

An additional goal is to define a PIM that is simple enough for the business people to understand. This will enable meaningful communications between the business system architects and the security architects by providing appropriate abstractions and a vocabulary. Thus, a person should not need to be a security domain expert to understand the concepts of the Access Control Platform Independent Model. For this reason, if the security community uses multiple terms for similar concepts, the choice of this work is to use the one that is most likely to be familiar to a business person, or to introduce a new term that can be mapped to the more technical security term during transformation.

It is also a goal of this work to unify existing access control mechanisms while providing abstractions that enable future innovation. Hence, the proposed model should be flexible enough to support the authorization requirements of the future infrastructures such as the Grid. To do this, a Model Driven Access Control must support an architecture where access decisions are architecturally separate from business logic. Therefore, this paper challenges the current trend of pushing knowledge of the underlying access control model into application logic when programmatic access control is required.

5. Model Driven Access Control

A model transformation occurs when models are refined and details are added for the purpose of focusing on a particular implementation technology or an aspect of the domain model. Model transformations are used to document different "levels of abstractions", "viewpoints" or "aspects" of an information system. Models that comply with a specific meta-model may utilize generative techniques for the transformations; leveraging information that the generator knows regarding the target implementation platform and/or parameterizations provided by the software architect [4].

If Model Driven architecture is to reach it's full potential, the quality of service issues must have Platform Independent Models. For the security access control models, that means that an AC-PIM must be explored from which JAAS, RAD and SAML/XACML (and others) can be derived. This ensures that business people and software architects could utilize common vocabulary and syntax to define the access control architectures. This paper supports the development of an access control PIM expressed in UML that can be the catalyst for the unification of the existing access control PSMs.

The definition of this model is critical to support the development of MDA tools that generate security access control points as a part of the infrastructure that bridges technology platforms and hopefully will provide a catalyst for innovations in the access control standards that push much of the application level access control down into infrastructure containers and/or communications layers. Finally, the adoption of a common model will enable the migration of existing access control implementations to a more consistent access control infrastructure. Thus, it should be possible to standardize using the experience-based design patterns and the mappings that enable true Enterprise Security Services to be integrated at all levels of computing infrastructure (Security related design patterns for scalability which can be utilized as part of model transformation were discussed in earlier work [2]). Model Driven Security means allowing the security contracts to be modeled as a part of the component contract, thus enabling the security context to be managed end-to-end by the UniFrame infrastructure. This will enable manageable access control and auditing regardless of whether the system composition was statically or dynamically generated.

6. Proposed Platform Independent Model

There are five principles that this model provides in support of Model Driven Access Control:

1) Access Control Points should be identifiable via parameterization of the domain PIMs with a single model element.
2) Protected Resources should be identifiable via the parameterization of the domain PIMs with a single model element.
3) An access policy should be defined, developed and managed separately from the application business logic. Access policy rules and access policy models must be able to evolve without any modification of the business software.
4) The policy model (role-based, user-based, code-based…) should not be exposed to the business application developers. That is, the business logic should have absolutely no knowledge of the access control model utilized to make access decisions.
5) The access control platform independent model (AC-PIM) must provide abstractions that are (as much as possible) consistent with the existing commonly used Platform Specific Models for access control. If not, it will not be acceptable to the user or vendor community and the work to produce it will be purely academic with no long-term impact.

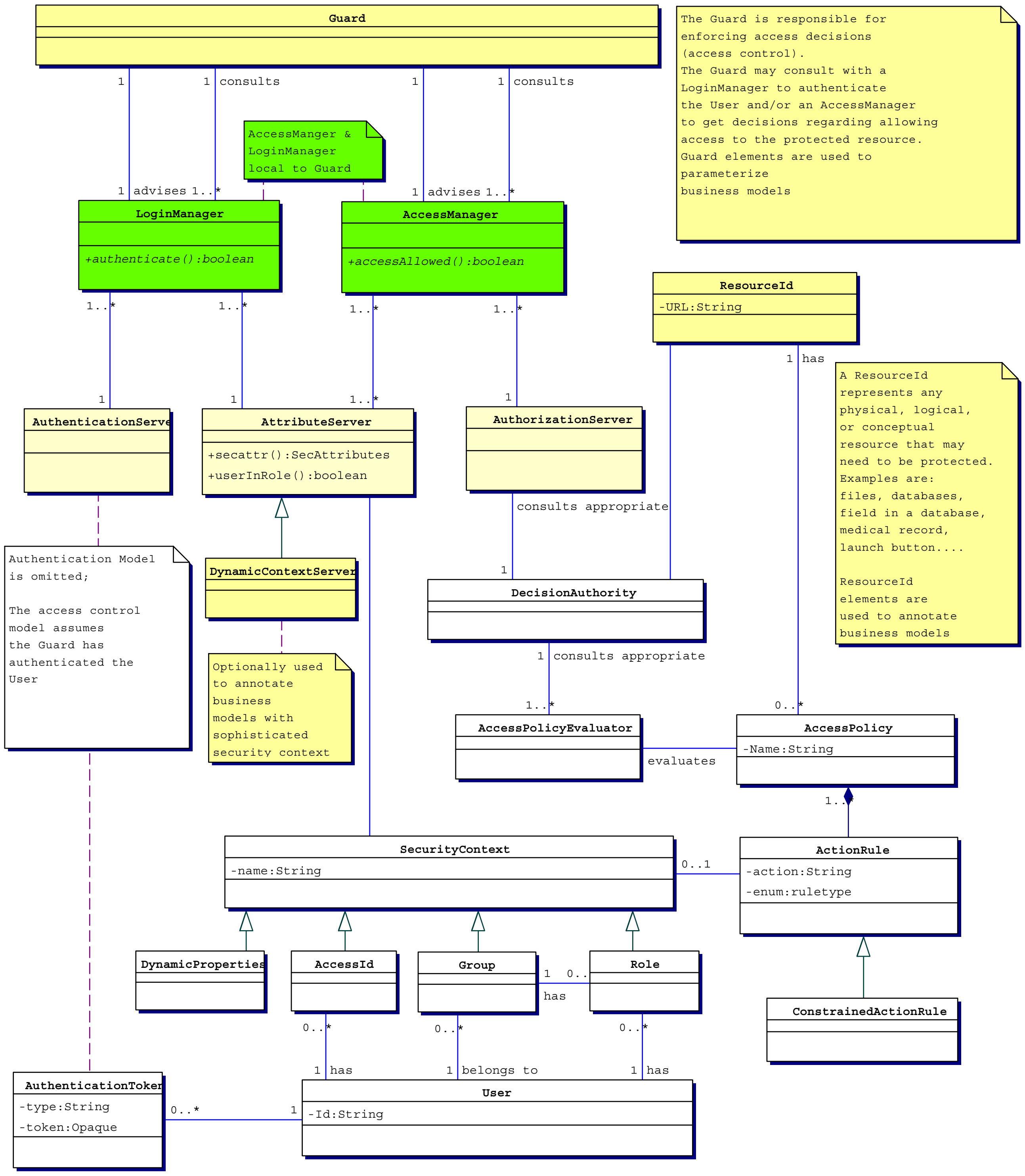
Guard
1
1 consults
1
1 consults
The Guard is responsible for enforcing access decisions (access control). The Guard may consult with a LoginManager to authenticate the User and/or an AccessManager to get decisions regarding allowing access to the protected resource. Guard elements are used to parameterize business models
AccessManger & LoginManager local to Guard
1 advises 1..*
1 advises 1..*
LoginManager
+authenticate():boolean
AccessManager
+accessAllowed():boolean
ResourceId
-URL:String
1 has
A ResourceId represents any physical, logical, or conceptual resource that may need to be protected. Examples are: files, databases, field in a database, medical record, launch button....
ResourceId elements are used to annotate business models
1..*
1..*
1..*
1..*
1
1
1..*
1
AuthenticationServe
AttributeServer
+secattr():SecAttributes
+userInRole():boolean
AuthorizationServer
consults appropriate
Authentication Model is omitted;
The access control model assumes the Guard has authenticated the User
DynamicContextServer
Optionally used to annotate business models with sophisticated security context
1
DecisionAuthority
1 consults appropriate
1..*
0..*
AccessPolicyEvaluator
evaluates
AccessPolicy
-Name:String
1..*
SecurityContext
-name:String
0..1
ActionRule
-action:String
-enum:ruletype
DynamicProperties
AccessId
Group
1 0..
has
Role
ConstrainedActionRule
0..*
0..*
0..*
1 has
1 belongs to
1 has
AuthenticationToken
-type:String
-token:Opaque
0..*
1
User
-Id:String

Figure 1 – Platform Independent Model for Access Control (AC-PIM)

To support these principles, we must explore the minimal knowledge that must be provided via parameterization during modeling. This includes:

1. Identification of Resources that might require protection (note – the architect may not know if a resource is protected and is only giving it an identifier so that the security policy can be defined at some future time).
2. Identification of the points within the application architecture where the access control checks should be made.
3. Identification of the application specific context /attribute information that might be needed at the point where an access control check is made (for example, withdrawal amount or credit limit or a relationship between a requestor and the information being requested).

The proposed AC-PIM model is shown in Figure 1. Although additional details related to the semantics and division of responsibility will need to be finalized in future work, this paper presents the initial design for the Access Control Platform Independent Model that will be used by the UniFrame team to progress unification of heterogeneous access control solutions. This model will evolve with that work. The modeling elements that represent the information that must be defined as parameterization of the domain model as discussed above are:

1. The PIM modeling element (object) that represents the resource and manages the identity is **ResourceId**. Note that there is not a model element for the protected resource as it is outside the scope of the AC-PIM. It is only present "by reference" in the model.
2. The PIM modeling element (object) that represents the access control points is a **Guard**.
3. The PIM modeling element (object) that supports a dynamic use of the context information in making the access decisions is a **DynamicContextServer**.

A ResourceId may represent any physical, logical or conceptual resource or a group of resources. For example, a ResourceId may represent a panel or button on a GUI, a feature of a cell phone, an operating system, an individual machine, an instance of a field in a file or database, an entire file or database, a C method invocation, an RMI or CORBA operation on an object, a process or application, or a concept such as "emergency room patient" or "psychiatric record". It is this "id" that represents the resource and is the target when requesting access decisions. The actual "resource" is stewarded by an application (or a real person in the case where it is a physical resource such as an x-ray film).

A Guard is inserted at every point within the application (or infrastructure) architecture where access control checks should be made and enforced. The decision regarding where to insert one or more Guard(s) will be made during the parameterization of the domain model prior to transformation to a platform specific technology for the components or system composition. Some technology platforms will provide Guards as part of the infrastructure (thus their meta-models already include one or more Guards). Others will require that Guards be manually added. Guards may be implemented by the operating system(s), messaging systems, infrastructure services, middleware, component containers, software components or applications. The Guard is responsible for ensuring that the user is authentic – that they are indeed who they claim to be (the authentication process is not covered by this paper – it is expected that any well accepted and popular methods of authentication would suffice) and that the user is authorized to access the protected resource. A Guard is typically a software piece protecting an electronic resource, but could also be a person who acts on the advice of software. The most important fact to understand is that it is the Guard and not the authorization service that is responsible for enforcement of the access control.

The ResourceId and Guard are the objects that will always be required to parameterize both domain models as developed by the business system architect, and the infrastructure component models used in the model transformation. The domain models must identify the resources that need to be protected and they must identify the points within the domain model where Guards should be activated. This parameterization will be utilized as a part of the model transformation to ensure that the access control checks are placed into the system at the appropriate points. In a similar way, infrastructure that enables dynamic system compositions (via the Web for example) must also support the dynamic identification of the protected resources and the insertion of Guards to protect them.

It is also anticipated that that some sophisticated business applications will require the ability to plug-in custom context servers that augment the infrastructure provided security services with an application specific security context. This plug-in ability is a requirement of an implementation of a compliant access control system. The DynamicContextServer provides the interface to support this feature. Architecturally this context server remains separate from the business logic and could be developed independently.

For the purposes of enabling Model Driven Access Control (via MDA tools), the remainder of the access control PIM shown in Figure 1 does not need to be exposed. We are, however, suggesting that the full model should be progressed as a mechanism for understanding and unifying the behavior of existing access control models and providing a consistent model that can be used as a basis of a reference model and vocabulary for future access control model evolution. For that reason, we define the full model in this paper. Now we will explore the architecture of the Access Control Platform Independent Model and how it supports the principles we outlined.

Consistent with most of the access control systems of today, a User may be either a person or a software component. The User is simply the requestor of access to a protected resource. When a user makes such a request, a Guard makes and enforces an access decision regarding whether to allow the User to access the protected resource that may be information or application feature(s). A ResourceId represents this protected resource.

The Guard has access to a local LoginManager and AccessManager for consultation purposes. The LoginManager and AccessManager are inserted into the model as locality constrained objects that provide an architectural support for location transparency and to address the need for high performance access control solutions; for example an implementation may support caching of the information stored in shared repositories and consolidation of multiple sources of the security information. These objects also support a unified application-programming interface (API) for requesting security authentication and authorization. This provides a plug-in point for vendors to integrate their solutions into heterogeneous environments.

The LoginManager's responsibility is to provide advice to the Guard on whether or not the requestor is who they claim to be. To do this, it uses an Authentication Service. Authentication Services may utilize diverse authentication technology including userid/password, X509 certificates, ticket-based (Kerberos), or token-based authentication. We will not explore a common authentication model in this paper except to note that this technology is significantly more mature in terms of allowing pluggable authenticators than authorization servers. The LoginManager has access to SecurityContextServer(s) for obtaining additional security context information if required during the process of authentication.

The AccessManager's primary responsibility is to provide advice to the Guard on whether or not the User should be should be allowed access to the resource(s) identified by the ResourceId. The AccessManager has access to one or more AttributeServer(s) and one or more AuthorizationService(s).

For a typical initial access request, the Guard will consult with the LoginManager to get advice on whether or not to trust the identity of the requestor and then consult with the AccessManager to determine if an access should be allowed to the requested resource(s). Subsequent requests for the resources from the same resource (typically within the scope of a session) would result in consultation only with the AccessManager. The Guard may also decide to trust an authentication token obtained by another Guard, which has been made available to it, and omit the consultation process with the local LoginManager. In this case, it would only consult with a local AccessManager. It is also possible that the Guard may determine that the resource is not protected at all and simply allows access to the resource without further consultation with either manager. Of course, a Guard may be disabled or removed which will allow access to all users, or may be inserted with a policy that causes it to deny access to all users.

It is important to understand that a Guard has the authority to accept or reject the advice of LoginManager and/or AccessManager(s) - although in practice this is typically not a good idea. This provides the flexibility to insert custom Guards in locations where normal application access control mechanisms must be disabled (such as emergency rooms where the information normally required to make decisions may not be readily available) or to temporarily deny access to everyone without any modification of the access policy. It is also important to note that the Guard does not have any knowledge of the access policy model. For example, if the underlying policy is role based access control (RBAC), the guard is not aware of the roles that may be required to access a resource. The Guard simply asks if an access is allowed or not by consulting with the AuthorizationService. A single implementation of a guard may therefore be used with multiple underlying access control models. This also supports our principle that the policy model (role-based, user-based, code-based, etc.) should not be exposed to the business application developers (as Guards will be provided by the application developers in environments where tools are not yet available to generate them). This is in contrast with current J2EE/EJB and Web Server programmatic security facilities which expose the role-based access control model by forcing a Guard to know the roles that are required and call "EJBContext::isUserInRole" or "HTTPServletRequest:: isCallerInRole" to make access decisions. The insertion of a local AccessManager into these architectures removes this requirement by allowing the Guard to call "AccessManager::access_allowed".

The AuthorizationServer, DecisionAuthority or AccessPolicyEvaluator(s) requires the security context information. The AccessManager must have this context information before consulting with an AuthorizationServer. Examples of the security context are security attributes such as groups, roles or access ids. Other examples of the security context are dynamic properties (such as the current balance on a checking account) that may be necessary to make an authorization decision. Such dynamic properties would be provided via a DynamicContextServer. By placing this in the access control architecture, a common design pattern is created that maintains the separation of the application logic and the access control logic and allows access policy to be evolved separate from the underlying business logic.

The AuthorizationServer consults an Access DecisionAuthority whose role is to combine the access decisions made by the PolicyEvaluators where multiple policies are in effect. For example, there may be a legal policy and an administrative policy that disagree. It would be the AccessDecisionAuthority that would be responsible for resolving any such conflicts. Simple AccessDecisonAuthority's would require consensus; more complex authorities might understand precedence rules. A PolicyEvaluator is responsible for evaluation of access policy. A PolicyEvaluator typically can evaluate any policy that follows a particular policy model (for example, role-based, access control lists or clearance based). Alternatively there may be PolicyEvaluators created for particular domains such as legal policy or administrative policy.

In the AC-PIM, the concept of AccessPolicy is abstract. An AccessPolicy consists of one or more Rules that are constructed using the SecurityContext. An AccessPolicy is associated (by name) with a ResourceId. The reason for requiring that AccessPolicy is associated "by name" to the ResourceId is to enable maximum scalability. By using this indirection, a policy can be managed independently and associated with many different resources. This is an expansion of the design pattern that enabled RBAC to scale (which creates "roles" that are assigned to Users and creating policy based on roles instead of individual users). This is more scalable than current deployment descriptors that require a redefinition of access policy rules in the deployment descriptor for each protected resource.

7. Transformation of the AC-PIM to Existing Platform Models

Figure 2 provides an overview of the models and aspects that must be considered as we look at the transformation of the AC-PSM to an AC-PSM for three existing Platform Specific Models.

1. The OASIS Access control model as defined in Security Assertion Markup Language (SAML) and eXtensible Access Control Markup Language (XACML).
2. The OMG access control model as utilized by the Resource Access Decision Facility (RAD).
3. The Java access control model as defined in the Java Authentication and Authorization Service (JAAS).

XACML is an XML specification for expressing policies. This specification defines an XML schema for an extensible access control policy language. As a result, it defines a standard vocabulary for the domain of access control policy. SAML is an XML-based security standard for a protocol to exchange the authentication and authorization information. This specification also defines the syntax and semantics for the XML-encoded SAML assertions about authentication, attributes and authorization.

	AC-PIM	JAVA / J2EE JAAS (JAAS PSM)	OMG RAD (RAD PSM)	OASIS SAML/ XACML (XACML PSM)
Access Control Model	Policy Based	Principal Based Access Control	Policy Based Access Control	Policy Based Access Control
Model Language	UML	Java Interfaces	ISO IDL	W3C XML
Description of Access Decision Model supported	An AccessManager uses a DecisionAuthority to make a decisions based on input from AccessPolicyEvaluators that may evaluate multiple policies. Named Policy is associated with ResourceIds	An AccessController determines if the Subject associated with the AccessControlContext has the required permission. Principals associated with the subject are matched against an application's required roles and permissions for the action are checked.	An AccessDecisionObject is passed the ResourceName, the requested operation on the resource, and SecurityAttributes. It locates the PolicyEvaluators and DecisionCombinator. One or more policies may be evaluated and combined by the combinator. Named Policy is associated with ResouceIds.	A Policy Decision Point (PDP) uses AuthorizationPolicy (gathered via PolicyRetrievalPoint) and evaluates it and makes an access decision which is provided via an AuthorizationAssertion
Policy Format standard	Not defined - Policy is named and associated with resources "by name". Policy format is encapsulated and is not standardized	XML - Policy is defined via grant statements in deployment descriptors that grant Permissions for actions to Security Principals (user/role)	Not defined - Policy is named and associated with resources "by name". Policy format is encapsulated and is not standardized	Policy is expressed via XACML statements as AuthorizationAssertions XACML provides a policy exchange language (in XML)
Access Decision API provided	The Guard calls AccessManager:: access_allowed()	The client sets AccessControlContext by invoking operation via Subject.doAs	The client (serving as a Guard) calls AccessDecision:: access_allowed	
Alternative native API	AttributeServer:: userInRole	EJBContext:: IsUserInRole		HttpRequest:: isCallerInRole
Infrastructure Guards (access control points)	Inserted via business model parameterization and generated by MDA tools	J2EE Web and EJB Containers and J2EE Connectors use JAAS Subject.doAs api	CORBA Security Service / CSIv2 may offer guard via interceptors.	Web Container and Servlet container is a SAML PolicyEnforcementPoint (PEP)
Application Guards (access control points)	Guard may be inserted by developers at identified access decision points	Developer may insert access decision point via Subject.doAs – java runtime then is used as guard	Developer may create a guard that uses the AccessDecisionObject:: access_allowed to get decisions.	Developer may insert SAML PEP in Valves or Filters or application code.
What the application components must identify	Guard insertion points, ResourceIds; actions on Resource; Optionally a custom SecurityContextService	References to external resources accessed References to inter-component calls made Ids of all role names if isUserInRole is used	ResourceNames ; actions on Resource; Optionally: custom DynamicAttributeService and/or custom PolicyEvaluator	References to external resources accessed References to inter-component calls made Ids of all role names used in isCallerInRole

Figure 2 – Contrasting the AC-PIM with existing Platform Specific Model

Table 1 indicates the conceptually similar modeling elements that need to be explored in an AC-PIM to XACML Platform Specific Model transformation:

AC-PIM	XACML-PSM
Guard	Policy Enforcement Point (PEP)
AccessManager	
LoginManager	
AuthenticationServer	AuthenticationAuthority
AttributeServer	AttributeAuthority
AuthorizationServer	AuthorizationAuthority
DynamicContextServer	PolicyInformationPoint
DecisionAuthority	Policy Decision Point (PDP)
ResourceId	URI reference

Table 1: Transformation to Key XACML Elements

Java and OMG CORBA share a security specification for secure interoperability. Common Secure Interoperability Specification, Version 2 (CSIv2) [27] supports the protocol necessary for infrastructure and applications to obtain the security context information necessary to leverage the Resource Access Decision Facility (RAD) for fine-grain access control. RAD provides a uniform service to assist in implementing infrastructure or application level access control where the protected resources may be physical, logical, or conceptual or understood only within the context of the business application. RAD was designed for use in multiple technology environments and addresses the problems of enterprises who have access control policy that is defined by privacy and confidentiality legislation (such as healthcare, telecommunications, and finance) These domains demand more sophisticated access control policies than what can be provided by infrastructure security.

Table 2 indicates the conceptually similar modeling elements that need to be explored in an AC-PIM to RAD Platform Specific Model transformation:

AC-PIM	RAD-PSM
Guard	RAD client
AccessManager	
LoginManager	
AuthenticationServer	AuthenticationService
AttributeServer	SecurityContext
AuthorizationServer	AccessDecisionObject
DynamicContextServer	DynamicAttributeService
DecisionAuthority	DecisionCombinator
ResourceId	ResourceName

Table 2: Transformation to Key RAD Elements

The Java™ Authentication and Authorization Service (JAAS) is a package that enables services to authenticate and enforce access controls upon users. JAAS authorization extends the existing Java security architecture that uses a security policy to specify what access rights are granted to executing code. JAAS authorization augments the existing code-centric access controls with new user-centric access controls. Permissions can be granted based not just on what code is running but also on who is running it. Permissions can be granted in the policy to specific Principals.

Table 3 indicates the conceptually similar modeling elements that need to be explored in an AC-PIM to JAAS Platform Specific Model transformation:

AC-PIM	JAAS-PSM
Guard	SecurityManager
AccessManager	Subject
LoginManager	LoginContext
AuthenticationServer	LoginModule
AttributeServer	Subject
AuthorizationServer	AccessController
DynamicContextServer	
DecisionAuthority	
ResourceId	Resource-ref

Table 3: Transformation to Key JAAS Elements

8. Conclusion

This paper proposes that access control patterns (in the form of a platform independent model) be utilized as a part of the component architectures to simplify the task of generating middleware that assumes the responsibility for the access control decisions that previously were tedious (or near impossible) to protect without the involvement of the application logic and the application developer. It proposes a Platform Independent Model that can be leveraged in a Model Driven Approach. While the full definition and standardization of such a security model is beyond the scope of this research project, this initial investigation indicates that the development of such a model is feasible. We are hopeful that this research will lead to the standardization of an Access Control Platform Independent Model (AC-PIM) under the OMG MDA process.

9. Future Work

Future work is planned to examine the problems associated with the protection of the fine-grain features and information resources of a typical Web-enabled business application. The case study, to be created for this purpose, will examine a Web-enabled business application architecture including a Web tier, an application logic tier, and an enterprise resource tier. Since an application may require that access to a resource in the enterprise resource tier be controlled based on the credentials of the Web user, it will explore the authorization and access control issues related to managing the security context across a multi-tier environment. A model-based solution will be proposed for the UniFrame project.

This case study will be used to validate the AC-PIM by completing the semantics of the models and the transformation to existing models. The proof of concept will take a well-known application (the Java Blueprints Pet Store), expressing the Pet Store domain model in UML and parameterizing it with the AC-PIM by identifying ResourceIds and inserting Guards that conform to the AC-PIM. A transformation of the model to the associated PSM that includes access control will then be progressed. An analysis of this manual transformation will serve as the foundation of code generators that mechanize the process. A final goal is to progress a standard Platform Independent Model for Access Control within the OMG community that can be leveraged by Model Driven Architecture tools.

We are currently involved in research in the use of formal methods for quality of service analysis in component-based distributed computing [28] and would like to investigate how formal methods might be leveraged in the access control domain. In addition, we hope to define future research projects that collaborate with groups doing natural language research and experiment with natural language processing of the access control requirements [29]. We also hope to collaborate with research groups that are using Aspect Oriented computing in Model Driven Architecture projects to determine if weaving techniques can be used to introduce access control logic during model transformation and at system composition time.

10. References

[1] Rajeev R. Raje, Barrett Bryant, Mikhail Auguston, Andrew Olson, Carol Burt. 2001. *A Unified Approach for the Integration of Distributed Heterogeneous Software Components*, Proceedings of the 2001 Monterey Workshop on Engineering Automation for Software Intensive System Integration, pp: 109-119.

[2] Carol C. Burt, Barrett R. Bryant, Rajeev R. Raje, Andrew Olson. Mikhail Auguston. 2002. *Quality of Service (QoS) Standards for Model Driven Architecture.* Proceedings of the 2002 Southeastern Software Engineering Conference, pp. 521-529.

[3] Carol C. Burt, Barrett R. Bryant, Rajeev R. Raje, Andrew Olson, Mikhail Auguston. 2002. *Quality of Service Issues Related to Transforming Platform Independent Models to Platform Specific Models.* Proceedings of EDOC 2002, the 6th IEEE International Enterprise Distributed Object Computing Conference, pp 212-223.

[4] Object Management Group. *Model Driven Architecture Guide*. Technical Report. Document # omg/2003-05-01. Framingham, MA: Object Management Group. May 2003.

[5] ITU-T Recommendation X.812 (1995) | ISO/IEC 10181-3: 1995, *Information Technology -- Open Systems Interconnection -- Security Frameworks for Open Systems -- Access Control*.

[6]http:://www.globus.org.

[7] K. Keahey, V.Welch. 2002. *Fine-grain Authorization for Resource Management in the Grid Environment.* Proceedings of Grid2002 Workshop.

[8] http://www.oasis-open.org.

[9] OASIS. 2003. The XACML 1.0 Specification Set, available via http://www.oasis-open.org.

[10] OASIS. 2002. Security Assertion Markup Language version 1.0, available via http://www.oasis-open.org.

[11] http://www.omg.org.

[12] Object Management Group. 2001. *Resource Access Decision Facility.* formal/2001-04-01 (full specification) formal/2001-04-02 (OMG IDL). Available via http://www.omg.org/technology/documents/formal/omg_security.htm.

[13] Sun Microsystems. 2002. *Java^{TM} Authentication and Authorization Service (JAAS) is part of Java 2 Platform Enterprise Edition Specification v1.4*, Available via ftp from www.java.sun.com. Sun Microsystems.

[14] http://www.jcp.org.

[15] Java Community Process. 2002. *JSR 115- Java^{TM} Authorization Contract for Containers*. Available for download from http://www.jcp.org.

[16] http://www.microsoft.com.

[17] http://www.gotdotnet.com/team/clr/about_security.aspx.

[18] D. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn, R. Chandramouli, 2001. *A Proposed Standard for Role Based Access Control*, ACM Transactions on Information and System Security , vol. 4, no. 3.

[19] D.F. Ferraiolo and D.R. Kuhn 1992. *Role Based Access Control.* 15th National Computer Security Conference

[20] Numerous references are available at http://csrc.nist.gov/rbac/].

[21] M. Abrams, J. Heaney, O. King, L. J. LaPadula, M. Lazear, and I. Olson. 1991. *A Generalized Framework* ***for Access Control: Towards Prototyping the Orgcon Policy*****,** In Proceedings of National Computer Security Conference, pp. 257-266.

[22] Beznosov, Deng, Blakley, Burt, Barkley. 1999. *A Resource Access Decision Service for CORBA-based Distributed Systems*. ACSAC (Annual Computer Security Applications Conference).

[23] International Standards Organization (ISO). ISO-IEC 14772-2. IDL as standardized by the Object Management Group.

[24] World Wide Web Consortium (w3c). Extensible Markup Language (XML) is text format derived from SGML (ISO 8879). Available from www.w3c.org/XML.

[25] Sun Microsystems Blueprints program. Pet Store version 1.3.1_01 available for download from http://java.sun.com/blueprints.

[26] Sun Microsystems. *Designing Enterprise Applications with the J2EE Platform*. Chapter on Pet Store Security Architecture. available online from http://java.sun.com/blueprints

[27] Object Management Group. 2001. *Common Secure Interoperability version2 - Chapter 24 of CORBA/IIOP specification.* formal/2002-12-06. available via http://www.omg.org/technology/documents/formal/omg_security.htm.

[28] Chunmin Yang, Barrett R. Bryant, Carol C. Burt, Rajeev R. Raje, Andrew M. Olson, and Mikhail Auguston, 2003. *Formal Methods for Quality of Service Analysis in Component-Based Distributed Computing* to appear in Proceedings of IDPT 2003, the Seventh World Congress on IntegratedDesign and Process Technology.

[29] Chunmin Yang, Beum-Seuk Lee, Barrett R. Bryant, Carol C. Burt, Rajeev R. Raje, Andrew M. Olson, Mikhail Auguston. 2002. *Formal Specification of Non-Functional Aspects in Two-Level Grammar*, Proceedings of the UML 2002 Workshop on Component-Based Software Engineering and Modeling Non-Functional Aspects (SIVOES-MONA), http://www-verimag.imag.fr/SIVOES-MONA/uniframe.pdf.