Supporting Heterogeneous Middleware Security Policies in WebCom

Simon N. Foley*, Barry P. Mulcahy, Thomas B. Quillinan,
Meabh O'Connor, and John P. Morrison
Department of Computer Science,
University College, Cork, Ireland.
{s.foley,b.mulcahy,t.quillinan,j.morrison}@cs.ucc.ie

February 14, 2006

Abstract

With the growing interest in service-oriented architectures, achieving seamless interoperability between heterogeneous middleware technologies has become increasingly important. While much work investigating functional interoperability between different middleware architectures has been reported, little practical work has been done on providing a unified and/or interoperable view of security between the different approaches.

In this paper we describe how the Secure WebCom distributed architecture provides access control policy interoperability support between a number of middleware security architectures. Secure WebCom uses the KeyNote trust management system to help coordinate the trust relationships between the different middleware systems and their associated access control policies. Middleware authorisation policies can be encoded in terms of cryptographic certificates, and vice-versa. This provides a unified view of access control across heterogeneous middleware systems and also provides the basis for decentralised support of middleware access control policies.

KEYWORDS: Middleware; Security; Web Services; Role Based Access Control; Trust Management; Interoperability.

*Corresponding Author

1 Introduction

Interoperability between heterogeneous middleware technologies is typically achieved through the creation of Middleware to Middleware (M2M) bridges [16, 24]. Separate bridges are created for each combination of differing middleware. Utilising these different bridges to provide smooth interoperability is bridge-dependent and is difficult to generalise. While these bridges provide *functional* interoperability between different middleware architectures [15], little practical work has been achieved on providing a unified and/or interoperable view of *security* between the different approaches.

Providing a unified view of access control policy facilitates the specification and integration of heterogeneous middleware components across complex application systems. This view should address the issues of policy configuration, comprehension, migration and maintenance.

Policy Configuration. Different middleware architectures may use different processes for specifying access control policies; getting the right configuration can be challenging. An approach to specifying a global access control policy that can, in turn, be commissioned in a consistent manner across the supported range of middleware architectures is required.

Policy Comprehension. Given a collection of independent access control policies configured across a range of middleware systems, their synthesis into a single unified policy is required. This synthesis promotes ease of understanding of the current state of the overall system security configuration.

Policy Migration. With policy comprehension comes an ability to migrate policies from one middleware architecture to another. For example, the ability to comprehend a Microsoft® COM+/.NET authorisation policy and synthesise an equivalent policy for an Enterprise Java Beans server.

Policy Maintenance. Maintaining ongoing changes to access control policies must be properly coordinated across the different systems to ensure a consistent view. Changes to individual access control policies must be propagated across the system to ensure consistency; changes to the global policy must be coordinated with changes to the individual middleware access control policies.

Conventional middleware security architectures such as CORBA [2], Enterprise Java Beans (EJB) [28] and Microsoft COM+/.NET [18] are based on Role Based Access Controls (RBAC). While providing distributed client-server architectures, middleware security mechanisms are effectively based on centralised RBAC security policies. Access is determined either by centralised authorisation servers, or by (possibly replicated) policies at local clients; there is little opportunity to coordinate differing access-controls and

policies across different domains and regions. System administrators are responsible for managing user authorisations and ensuring proper coordination across potentially many different policies.

Trust management schemes [4–6, 9, 25] use public key authorisation certificates to specify delegation of authorisation between public keys and can be used to help decentralise authorisation policies. Trust Management is an approach to constructing and interpreting the trust relationships between public keys that are used to mediate security critical actions. Cryptographic credentials are used to specify delegation of authorisation among public keys.

Distributing these access control policies throughout the different Middleware technologies requires a bridge between the different systems. WebCom [21] is a distributed meta-computing architecture that provides interoperability between different middleware. WebCom applications are developed using a variety of middleware components; WebCom coordinates their execution across the appropriate client/servers, according to a condensed graph [19] that defines the execution sequencing rules for the application. WebCom also provides services for Grid middlewares [20]. Secure WebCom [14] uses Trust Management [4–6, 9, 25] to manage authorisation within WebCom. Cryptographic authorisation credentials are used to determine whether it is permitted for a middleware client/server to execute a component. While Secure WebCom uses a pluggable architecture to support a variety of trust management engines [22], we use KeyNote [4] in this paper to demonstrate our approach.

This paper is a revised version of the paper originally presented as [11]. In this paper we describe how Secure WebCom provides interoperable access control policies across different middleware environments. This support integrates the authorisation schemes provided by the underlying operating system, the middleware system, and trust management based authorisation provided by Secure WebCom. The approach provides a number of advantages, addressing the issues of policy configuration, comprehension, migration and maintenance. While conventional middleware security architectures typically rely on a centralised notion of authorisation, KeyNote certificates may be used to promote a decentralised approach.

The paper is organised as follows. Section 2 provides an interpretation of common middleware authorisation policies in terms of a simple Role Based Access Control Model (RBAC). An introduction to Trust Management and the KeyNote system in particular, are given in Section 3. Section 4 describes how KeyNote is integrated into WebCom and how interoperability between KeyNote and the underlying middleware access control policies is supported. WebCom facilitates the support of both trust management based policies

and underlying middleware policies, as described in Section 6. WebCom provides an environment for developing distributed applications built from different middleware components; Section 7 outlines how such applications can be developed when taking security into consideration.

2 Middleware RBAC Security

Role-based access control (RBAC) [27] is widely used to provide access control in database management systems, operating systems and Middleware architectures. In RBAC, access rights (permissions) are associated with roles, and users are members of these roles. When a user is assigned to a role, they gain all the permissions of that role in the system. This allows an organisation to model its security infrastructure along the lines of its business. For the purposes of this paper we make more concrete the conventional RBAC model of *Users*, *Roles* and *Permissions*, to include *Domains*.

- *Permission*: represent actions, capabilities, applications or any other active behaviour that can be "performed" and, to which, we intend to control authorisation.
- *Domain*: administrative boundaries that group permissions and manage their underlying resources. In general, domains do not intersect in their underlying permissions.
- *Role*: roles are logical groupings of permissions that reflect a particular task that can be assigned to some user. We assume that roles do intersect in their underlying permissions.
- *User*: include humans or any other entity that can be assigned a role.

An RBAC policy is defined in terms of the following relations.

$$\begin{aligned} RolePerm &: (Domain \times Role) \leftrightarrow Permission \\ RoleUser &: (Domain \times Role) \leftrightarrow User \end{aligned}$$

where $RolePerm((d, r), p)$ means that the role r (in domain d) holds permission p (on some object), and $RoleUser((d, r), u)$ means that user u is assigned to domain-role pair (d, r). Table 1 uses this model to provide a uniform interpretation of basic COM+, EJB and CORBA middleware RBAC policies.

Enterprise Java Beans (EJB) [28] EJB components encapsulate the business logic of Sun Microsystems Java 2 Enterprise Edition (J2EE) application model. Components reside in a bean container located on a server, running on a host machine. The combination of host, EJB server, and the relevant bean container JNDI [29] name provide the *domain*s of the policy. *Role*s are bean-specific in each container. *User*s exist globally in each EJB server, and as such can be members of roles in different domains. Users are unique to each EJB server, and are managed by that server. *Permission*s represent method calls that a role is permitted to make on an EJB object.

Microsoft COM+/.NET [18] COM objects can be anything from a simple software program right up to a complex program, such as Word. COM's RBAC model is an extension of the Windows security model and provides Windows NT *Domain*s, *Role*s unique to each *Domain*, and *Permission*s comprised of primitive COM permissions and object references. For the purposes of this paper, the set of primitive COM permissions used are `Launch`, `Access`, and `RunAs`.

Common Object Request Broker Architecture (CORBA) [2] The CORBA RBAC model consists of *Role*s, *User*s and *Permission*s. We consider a domain to be the name of the machine and the CORBA ORB server name, similar to the EJB system. Roles are assumed unique to each domain, and users can be members of one or many roles. Permissions relate to the method calls on objects of the given object type.

Example 1 A stock market trading application system ShareTrader manages the trading of shares. This system is configured across a trading manager's Windows workstation (domain mgmt) and a J2EE server (domain staff) that supports trading operations. The application system has operations setlimit, analyzerisk, pricedeal, and capturedeal. We have the role hierarchy Sales $<$ Trader $<$ TraderMgr, and Figure 1 provides a simple example of the RBAC policy.

In practice, policy attribute values are defined in terms of architecture specific details. For example, a share trader application running on a Java EJB server staffserver, port 1050, would have a domain specified as

"staffserver/1050/ShareTrader/ShareTraderEJB"

and offer java EJB permissions such as "captureDeal java.lang.String", and so forth. On the windows's workstation, policy attributes are encoded

in terms of the COM+ policy attributes. For example, the permission ("ref:A4C....",Launch) indicates the authority to launch the COM component referenced by "A4C....". If a middleware operation is referred to within a WebCom application then it will also have a local WebCom name. For example, a share trader application that is managed by Secure WebCom would have sample WebCom permission name "webcom.nodes.ShareTrader.PriceDealOp". In practice, Secure WebCom uses a SDSI-like naming system to provide consistent naming across the many different middleware platforms [22]. △

3 Trust Management

Cryptographic delegation certificates/credentials specify delegation of authorisation between public keys. When a request from an untrusted principle (key) is made to a networked application to execute a particular action, then, authentication notwithstanding, the application must determine whether the key(s) that made the request is authorised. Trust Management approaches such as KeyNote [4] and SPKI [9] provide assistance to applications in making these decisions. Trust Management facilitates a decentralised approach: authorisation may be determined without having to consult some central authorisation server, and users may choose to further delegate their authority without having to refer to a Central Authority.

Authorisation is determined via digitally signed public key credentials that bind public keys to the authorisation to perform various actions. For example, Sally may hold a credential, signed by her manager's private key, binding her public key to her authorisation to capture deals up to a value of $100.00. Sally's public key signs an order request to the purchasing application; her credential provides proof of authorisation.

In practice, authorisation is achieved by a collection of credentials that exhibit the necessary trust relationships between their keys. For example, we may trust Sally's public key for orders up to $100.00, if her manager's public key is trusted to delegate orders up to $100.00 (or more), and so forth along a delegation chain that ends in a key that is known to be appropriately trusted. Given a policy (public keys, trusted in known ways), and a collection of credentials, a network application must determine whether a particular public key is authorised to request a particular operation.

KeyNote [3, 4] is a flexible trust management scheme that provides a simple credential notation for expressing both access control policies and delegation. A standard KeyNote Application Programming Interface is used by an application to make queries about whether security critical requests (to the

application) have authorisation or not. The formulation and management of security policies and credentials are separate from the application, making it straightforward to support trust management policies across different applications. KeyNote has been used to provide trust management for a number of applications including Secure WebCom [14], Grid Administration [23], managing Security Associations in IPSec [7] and web servers [1].

Example 2 Suppose that the share trader application (Example 1) is implemented on a single server and uses KeyNote to determine whether user requests (to execute application operations) are authorised. Trading Manager Mandy is authorised for all actions; this is specified by the following ad-hoc[1] KeyNote credential.

```
Authorizer: POLICY
licensees:  "KMandy"
Conditions: app_domain=="ShareTrader" &&
    (oper=="setlimit" || oper=="analyzerisk"
||   oper=="pricedeal"|| oper=="capturedeal");
```

This *policy* credential defines the conditions under which requests from the licensee key `KMandy` may be trusted by the application `ShareTrader`. These conditions are defined using a C like expression syntax in terms of the *action attributes*. In this example, attributes `app_domain` and `oper` are used characterise the circumstances of a request. Note that for the purposes of illustration in this paper we use short simple names to represent public keys rather than encoding them as proper cryptographic keys.

The owner of public key `KMandy` has the authority to delegate this trust to other keys and does so by signing the following credential for a salesperson who owns public key `KSally`.

```
Authorizer: "KMandy"
licensees:  "KSally"
Conditions: app_domain=="ShareTrader" &&
    (oper=="pricedeal || oper=="capturedeal");
```

In signing this credential, authoriser `KMandy` delegates authority for pricing and capturing deals to the key `KSally`.

When `KSally` requests to capture a deal (sending a request signed by `KSally`) then the application queries KeyNote with authoriser `KSally`, action attributes $\{$`app_domain` $\leftarrow$ `"ShareTrader"`, `oper` $\leftarrow$ `"capturedeal"`$\}$, the policy credential above, and a set of signed credentials provided by Sally.

[1] Section 4 will consider the encoding of RBAC policies in terms of KeyNote credentials.

KeyNote confirms that this key is authorised since, by default (policy), we trust `KMandy` for all share trader operations and `KMandy` has delegated some of this trust to `KSally`, by virtue of signing the credential. △

4 Secure WebCom

Secure WebCom [14, 21] is a distributed secure and fault-tolerant architecture that is used to coordinate the distributed execution of middleware components across a network. A Secure WebCom environment [14, 22] supports a variety of pluggable trust management engines, including KeyNote [4] and SPKI/SDSI [9]. Figure 2 outlines the role of trust management in Secure WebCom. The WebCom master uses the client credentials to determine what operations it may schedule to them. Similarly, each WebCom client uses the master's credentials to determine whether the master is authorised to schedule the operation. WebCom clients may also act as masters to other WebCom clients.

In practice, a WebCom master/client is launched by, and runs as, a user of the system. While a WebCom master might run as a special user on a network server, individual users can run copies of WebCom clients on their local workstations. Instances of WebCom masters and/or clients use their owner's X509 certificates to achieve secure authentic connections via SSL/TLS. Note that the identities provided by these certificates are not used by Secure WebCom; rather they provide a practical way to link messages that are sent over secure (SSL) connections to public keys (X509) that are authorised by KeyNote credentials. Secure WebCom uses Trust Management credentials to determine the authorisation of X509 authenticated SSL connections

5 Supporting RBAC Policies in Secure WebCom

RBAC-like policies can be encoded in terms of equivalent cryptographic certificates/policies [17, 25]. In addition to supporting ad-hoc KeyNote policies, Secure WebCom supports middleware RBAC-like security policies by effectively encoding the corresponding middleware $RolePerm$ and $RoleUser$ relationships (from Section 2) within KeyNote authorisation credentials. This is unlike [17] which integrates authorisation certificates as part of the lower-level middleware system. Secure WebCom uses KeyNote to determine whether it is safe to execute a middleware component.

A Secure WebCom environment can automatically convert middleware RBAC policies to their equivalent KeyNote policies/credentials, and vice-versa. This provides a high degree of policy interoperability, between the middleware and trust management layer, and also within the different middlewares. In addition to providing a uniform way of specifying RBAC policies for different middleware systems, it also becomes possible to enforce standardised RBAC middleware policies across middleware systems that do not have a configured RBAC policy.

There are a variety of approaches to supporting roles in KeyNote. Encoding the fixed relationships from the *RolePerm* table as a single KeyNote credential provides a straightforward representation of the RBAC policy. Individual credentials are then issued, associating users to roles.

Example 3 The `ShareTrader` Domain-Role-Permission table can be encoded as the following policy credential.

```
Authorizer: POLICY
Licencees: "Kwebcom"
Conditions: app_domain="ShareTrader" &&
  (Domain=="mgmt"&&(role=="TraderMgr") ->
    (perm=="setlimit"||perm=="analyzerisk"||...);
   ....
  (Domain=="staff"&&(role=="sales") ->
    (perm=="pricedeal"||perm=="capturedeal");
```

This specifies that a WebCom administration key `Kwebcom` is authorised to administer rights in connection with this policy. This could be used as the policy credential for a WebCom master that is under the control of the owner of `KWebCom`. The credential

```
Authorizer: "Kwebcom"
Licencee:   "Kjoe"
Condition: app_domain=="ShareTrader" &&
           role=="TraderMgr";
```

authorises public key `Kjoe` as a share trader. This means, for example, that Joe may make `analyzerisk` requests from a WebCom client executing on his local workstation. △

The above approach promotes a more centralised policy administration, with the WebCom environment (administrator) managing delegation and is comparable to the conventional middleware approach.

Alternatively, the Domain-Role-Permission table can be decentralised and spread across a number of credentials and additional authorisations and

role memberships delegated to other keys. A common strategy is to represent roles (from domains) in terms of public keys; delegation is used to create the role-permission and role-user relationships. In practice, roles are best supported using SDSI-like local names [25], however, we can approximate the role membership effect in KeyNote as follows.

Example 4 Public keys `KRtrader` and `KRsales`, etc., are used to represent roles. Credentials associate authorisations to the roles. For example,

```
Authorizer: KRtrader
Licencee:   KRsales
Condition: app_domain=="ShareTrader" &&
    perm=="pricedeal"||perm=="capturedeal";
```

Sally is assigned this role using a credential, signed by `KRtrader`, authorising `Ksally`. In practice, if Joe is a member of the `KRtrader` and is permitted to further delegate the associated permissions, then Joe could authorise Sally to be in the `KRsales` role. △

A disadvantage of this more flexible and decentralised approach is that in giving administration authority to individual users it provides only limited control of how these users subsequently delegate their authority; trading manager Mandy can decide to directly authorise salesperson Sally to setlimit, regardless of the intended role hierarchy. In [10] we describe how distributed workflow rules supported by WebCom are used to place constraints on the delegation actions of such users.

5.1 Policy Configuration

It is not necessary to rely on just the access control (KeyNote) mechanism of Secure WebCom; the underlying middleware and operating system also provide RBAC security mediation. In this case, it is necessary to translate the specified KeyNote RBAC policy into its equivalent middleware RBAC security configuration.

A WebCom environment provides an administrator-privileged service called KeyStar that accepts network requests to update the underlying middleware security policy. The KeyStar service accepts a request that is authenticated as coming from a user/WebCom with public X509 key K. This request may be to AddRole(R), AddUser(U), AddUsertoRole(U,R), or AssignPermtoRole(R,P), where U is the userid of a middleware user, R a role and P a permission. KeyStar uses the credentials provided with the request to

check whether the requesting key K is authorised for this action, and if so, then it updates the middleware policy to reflect the request.

In practice, KeyStar is implemented as superclass, providing an uniform interface and carrying out the Trust Management compliance check. This superclass is extended when implementing the middleware-specific policy update operations. Secure WebCom currently provides KeyBean and KeyCOM services which extend KeyStar to support COM+ RBAC and EJB RBAC policy update, respectively. A Keynix service provides support for standard Unix policies, with user-groups approximating roles.

For example, Figure 3 outlines how a user, currently registered only in Domain B, is integrated into a COM+ RBAC policy within Domain A. The KeyCOM service of WebCom accepts a policy update request (plus KeyNote credentials) and if valid it updates the security policy in the COM+ Catalogue (middleware/Windows RBAC policy) with the equivalent authorisation. KeyCOM acts, in effect, as an automated Windows/COM administrator, processing client authorisation requests, while the KeyNote cryptographic credentials facilitate users in delegating authorisation without requiring assistance of non-automated (that is, human) administrators.

5.2 Policy Comprehension

It is also useful to translate middleware RBAC policies into equivalent (WebCom) KeyNote RBAC policies. In this case the middleware RBAC policy can alternatively be enforced by Secure WebCom. A middleware RBAC policy comprised of relations $RolePerm$ and $RoleUser$ is converted into KeyNote credentials as follows. This translation process uses Table 1 to interpret the middleware RBAC policy in terms of domains, roles, permissions and users.

- The $RolePerm$ is encoded as a KeyNote policy credential that authorises the WebCom Key as administrator for the middleware system's RBAC policy and is authorised for the given values over attributes domain, role, and Permission. Example 3 illustrates this translation with the KeyNote policy credential encoding the $RolePerm$ table.

- For each user in the $RoleUser$ table, a KeyNote credential is generated, and signed by the WebCom administration key (owner of the given middleware system), authorising the user public-key to be a member of the corresponding roles from $RoleUser$. The credential for `Kjoe` illustrates this in Example 3.

In practice, the generation of KeyNote credentials from middleware policies is provided by WebCom middleware *interrogators*. These extract information on avaliable middleware components, including policy information, for use within the WebCom IDE (Section 7). The RBAC policy obtained by the interrogators is also translated into corresponding KeyNote credentials. Secure WebCom currently supports COM+, EJB and CORBA interrogation.

5.3 Policy Migration

Migration of existing policies from one middleware system to another is also possible. This interoperability of disparate access control policies allows, for example, a new system to be configured with the same policy as an existing system.

Figure 4 illustrates this interoperability with a sample configuration. System Z is a WebCom Server, while systems X, Y and Z are WebCom clients. System Z security relies on the Windows (W) operating system, COM middleware and KeyNote trust management. These systems could have independent policies, or might require a more homogeneous policy across the different platforms. A WebCom client running on Windows with COM middleware security policy inter-operates with the server. If required, the KeyNote RBAC credentials held by users of System Z can be used to update the COM+ catalogue of System Y. On the other hand, the COM middleware RBAC policy on System Y can be translated to equivalent KeyNote credentials and these, in turn, used by System W which does not have a middleware security mechanism. In addition, if System Y was a legacy system under migration to System X, then the KeyNote credentials generated from the legacy COM policy can be used to automatically configure the replacement EJB RBAC policy.

Policy migration is currently supported in Secure WebCom via policy comprehension (using policy interrogators to extract equivalent credentials from middleware policies) and policy configuration (using KeyStar authorisation services to update middleware policies). This is a straightforward translation when the middleware names (users, roles, etc.) referred to are identical. However, it requires administrator intervention when different names are to be considered 'equivalent' during a migration. For example, it might be decided that the COM role TraderMgr in domain mgmt is considered equivalent to an EJB role TraderMgr in domain staff. Interrogated policies must be modified to reflect these equivalences before the KeyStar service is used. For example, replacing term `(Domain=="mgmt")`

by `(Domain=="mgmt"||Domain=="staff")` in the KeyNote policy in Example 3. Credential similarity metrics [12] can be used to provide control over more flexible equivalences between names.

5.4 Policy Maintenance

The maintenance of a consistent global policy across the different heterogeneous middlewares is important for the overall security of the system. Making changes to the underlying middleware access control policies can lead to inconsistencies between the authorisation of principals on different systems.

Maintaining both middleware and trust management policies is an important aspect of the Secure WebCom system. For example, if a new employee is to be added to the existing policy, changes must be propagated to all the relevant heterogeneous system policies. It is possible to add entries to the underlying middleware access control policies and then utilise the translation services to propagate these policy changes to the other systems. However, we recommend changing the trust management policy to reflect required changes in the system. This enables the changes to be propagated where necessary, while maintaining the consistency of the overall access control policy. Making the changes to the trust management policy has additional benefits. For example, a manager can delegate authority to a new employee without requiring detailed knowledge of the underlying systems in use.

Continuing the Share Trader example, the manager can achieve this by the creation of new credentials, assigning the employee the roles required. The administrative translation services can update the middleware policies to permit the new employee to use the services appropriate to their function. Maintaining the policy in this fashion has the additional benefit that it provides a high-level view of the overall policy of the organisation. There are advantages to this approach over existing work, such as [17]. For example, we have the ability to both abstract existing access control policies into Trust Management policies and conversely enforce portions of Trust Management policies in terms of middleware RBAC policies.

6 Stacked Authorisation

Using the trust management architecture in Secure WebCom requires that the WebCom environment be trusted in the sense that part of the security mediation (authorisation) is performed by the WebCom environment

and not the underlying operating system. An advantage of this approach is that, since it is independent of the security architecture of the underlying system then it provides a better opportunity for interoperation between heterogeneous platforms that run the WebCom environment. However, since it does not rely on the underlying operating system/middleware authorisation mechanisms, a result is that it increases the software in the trusted computing base.

In the case where the WebCom system is not trusted by the operating system, the access control policy of the middleware applications may be enforced. This implies that a choice between enforcing the middleware and WebCom security policies must be made.

We address this property by considering how the security mechanisms of the underlying middleware/operating system can be used to provide the basis of security mediation and form a part of the overall WebCom security architecture. This provides a stack of security layers, as depicted in Figure 5. Note that the Level 3 security corresponds to mechanisms encoded within the condensed graph that is used to coordinate the application components. It is used to implement application level workflow security, for example [13], and is not considered in this paper.

These stacked layers of secure WebCom are 'pluggable' in the sense of [26]; for example, in the absence of CORBASec support for a particular ORB, a WebCom environment could be configured so that authorisation is based only on a combination of KeyNote (trust management) and underlying operating system policy.

Using this stackable architecture, applications can be developed that use trust management policies for new components and the middleware security architectures for existing security mediations. This has the benefit of using legacy code and policies, minimising inconsistencies in the conversion of applications.

7 Secure Application Development

A distributed WebCom application is constructed as a condensed graph [19] of components using an Integrated Development Environment (IDE). The IDE provides a platform to build distributed middleware applications based on existing middleware business logic and access control policies. Through this approach, programmers can easily build secure and complex distributed applications from legacy components.

To incorporate the existing middleware components as part of a Web-

Com application, existing middleware services are interrogated and made available to application developers through the use of a component palette on the WebCom IDE. Interrogation is achieved using a WebCom plugin specific to the middleware service in question. The interrogated RBAC policy (Section 5.2) is also available within the IDE as an additional palette (Figure 6). The IDE analyses the middleware component currently highlighted, and determines which combinations of domain, role and user is suitably authorised (holds permissions) to execute the selected component.

The programmer may specify any valid combination of domain, role and user for a component to execute under. The WebCom scheduler ensures components are scheduled to the specified domain, role, and user. A partial specification is also supported, for example, allowing the programmer to specify a domain and role for a given component, in which case it will be scheduled to any authorised user in the specified domain and role.

8 Discussion and Conclusion

We have described how interoperable access control is supported for the different middleware environments that are supported by WebCom. With this approach, authorisation certificates provide a decentralised approach for authorisation. We also described how existing middleware RBAC policies can be encoded in terms of KeyNote credentials and vice-versa. This provides a unified view of access control of a heterogeneous middleware application system, and also provides the basis for the support of centralised and decentralised RBAC middleware security.

Secure WebCom [14, 22] is a distributed secure and fault-tolerant architecture that is used to coordinate the distributed execution of middleware components across networks. We use WebCom in this paper to provide an infrastructure to support our approach to middleware RBAC interoperability. The interested reader is referred to [14, 21, 22] for further information on the underlying WebCom architecture.

Work in the area of trust management has focused on replacing existing security models with Trust Management systems. The emphasis in our research has been more of one to use trust management in a practical setting where it is not feasible to re-engineer legacy application systems. We have developed a system that uses the existing RBAC systems, and uses Trust Management to create more integrated policies.

Acknowledgements

The authors would like to thank the anonymous referees for their helpful comments on this paper. The authors would also like to thank the members of the Centre for Unified Computing in UCC, without whose support this work would not have been possible. This work was supported in part by the Informatics Research Initiative of Enterprise Ireland.

References

[1] Apache-ssl release version 1.3.6/1.36. Open source software distribution. `http://www.apache.org`.

[2] G.R. Blakley. *Corba Security. An Introduction to Safe Computing with Objects.* Object Technology Series. Addison-Wesley, 2000.

[3] M. Blaze. Using the KeyNote trust management system. `http://www.crypto.com/trustmgt`, December 1999.

[4] M Blaze et al. The keynote trust-management system version 2. September 1999. Internet Request For Comments: 2704, IETF.

[5] M Blaze et al. The role of trust management in distributed systems security. In *Secure Internet Programming: Issues in Distributed and Mobile Object Systems*. Springer-Verlag Lecture Notes in Computer Science, 1999.

[6] M Blaze, J Feigenbaum, and J Lacy. Decentralized trust management. In *Proceedings of the Symposium on Security and Privacy*. IEEE Computer Society Press, 1996.

[7] M. Blaze, J. Ioannidis, and A. Keromytis, Trust Management for IPsec, In *Proceedings of Network and Distributed System Security Symposium*, The Internet Society, 2001,

[8] M. Blaze, J. Ioannidis, and A.D. Keromytis. Trust management and network layer security protocols. In *Security Protocols International Workshop*. Springer Verlag LNCS, 1999.

[9] C. Ellison et al. SPKI certificate theory. September 1999. Internet Request for Comments: 2693, IETF.

[10] S.N. Foley, B.P. Mulcahy, and T.B. Quillinan. Dynamic administrative coalitions with webcom *DAC*. In *WeB2004 The Third Workshop on e-Business*, Washington D.C., USA, December 2004.

[11] S.N. Foley, T.B. Quillinan, M. O'Connor, B.P. Mulcahy, and J.P. Morrison. A framework for heterogeneous middleware security. In *Proceedings of the 13th International Heterogeneous Computing Workshop*, Santa Fe, New Mexico, USA., April 2004. IPDPS.

[12] S.N. Foley. Supporting imprecise delegation in keynote. In *Proceedings of 10th International Security Protocols Workshop*, April 2002, Springer Verlag LNCS 2845.

[13] S.N. Foley and J.P Morrison. Computational paradigms and protection. In *ACM New Computer Security Paradigms*, Cloudcroft, NM, USA, 2001. ACM Press.

[14] S.N. Foley, T.B. Quillinan, and J.P. Morrison. Secure component distribution using webcom. In *Proceeding of the 17th International Conference on Information Security (IFIP/SEC 2002)*, Cairo, Egypt, May 2002.

[15] R. Geraghty, S. Joyce, T. Moriarty, G. Noone, and Sean Joyce. *COM-CORBA Interoperability*. Number ISBN: 0-130-96277-5. Prentice Hall PTR, 1998.

[16] J. Hugues, F. Kordon, L. Pautet, and T. Quinot. A case study of middleware to middleware: Mom and orb interoperability. In *Proceedings of the 4th International Symposium on Distributed Objects and Applications (DOA'02)*, Irvine, CA, USA, October 2002. University of California, Irvine.

[17] T. Lampinen. Using SPKI certificates for authorization in CORBA based distributed object-oriented systems. In *4th Nordic Workshop on Secure IT systems (NordSec '99)*, pages 61–81, Kista, Sweden, November 1999.

[18] Microsoft Corporation. *Microsoft Platform SDK. The COM Library. Microsoft Developer Network.*, 0.9 edition, October 1995. http://www.msdn.microsoft.com.

[19] J.P. Morrison. *Condensed Graphs: Unifying Availability-Driven, Coercion-Driven and Control-Driven Computing*. PhD thesis, Eindhoven, 1996.

[20] J.P. Morrison et al. WebCom-G: Grid enabled metacomputing. *Neural, Scientific and Parallel Computations Journal.*, Vol. 12(3), PP. 419-438, September, 2004

[21] J.P. Morrison, D.A. Power, and J.J. Kennedy. A Condensed Graphs Engine to Drive Metacomputing. Proceedings of the international conference on parallel and distributed processing techniques and applications (PDPTA '99), Las Vagas, Nevada, June 28 - July1, 1999.

[22] T.B. Quillinan and S.N. Foley. Security in WebCom: Addressing naming issues for a web services architecture. In *Proceedings of the 2004 ACM Workshop on Secure Web Services (SWS).*, Washington D.C., USA., October 2004. ACM.

[23] T.B. Quillinan, B.C. Clayton, and S.N. Foley. GridAdmin: Decentralising grid administration using trust management. In *Proceedings of the Third International Symposium on Parallel and Distributed Computing (ISPDC04)*, Cork, Ireland, July 2004. *To Appear.*

[24] T. Quinot, F. Kordon, and L. Pautet. From functional to architectural analysis of a middleware supporting interoperability across heterogeneous distribution models. In *Proceedings of the 3rd International Symposium on Distributed Objects and Applications (DOA'01)*. IEEE Computer Society Press, September 2001.

[25] R Rivest and B Lampson. SDSI - a simple distributed security infrastructure. In *DIMACS Workshop on Trust Management in Networks*, 1996.

[26] V. Samar and R. Schemers. Unified login with pluggable authentication modules (PAM). Request for Comments 86.0, Open Software Foundation, October 1995.

[27] R Sandhu et al. Role based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[28] Sun Microsystems. *Enterprise JavaBeans(tm) Specification, Version 2.1*, June 2003. http://java.sun.com/products/ejb/docs.html.

[29] Sun Microsystems Inc. *Java Naming and Directory Interface*, 1.2 edition. http://java.sun.com/products/jndi/.

Domain	*Role*	*Permission*
mgmt, staff	TraderMgr	setlimit, analyzerisk, pricedeal, capturedeal
staff	Trader	analyzerisk, pricedeal, capturedeal
staff	sales	pricedeal, capturedeal

Domain	*Role*	*User*
mgmt	TraderMgr	Mandy
staff	Trader	Joe
staff	Sales	Sally

Figure 1: RBAC relations for ShareTrader System

Type	*Domain*	*Role*	*User*	*Permission*
EJB	Combination of Host, EJB Server, relevant bean container.	Application Specific for each server.	Exist globally on each server, can be members of different roles.	Method calls (of an object type) that roles are permitted to make.
COM	Windows NT Domains.	Unique to Domains.	Windows Users. Unique to each Domain.	Considering only `Launch`, `Access` and `RunAs` in this paper.
CORBA	Machine name and ORB server name.	Unique to Domains.	Can be members of different roles, unique to each server.	Relate to method calls on objects of the given object type.

Table 1: Interpretation of Middleware RBAC Models

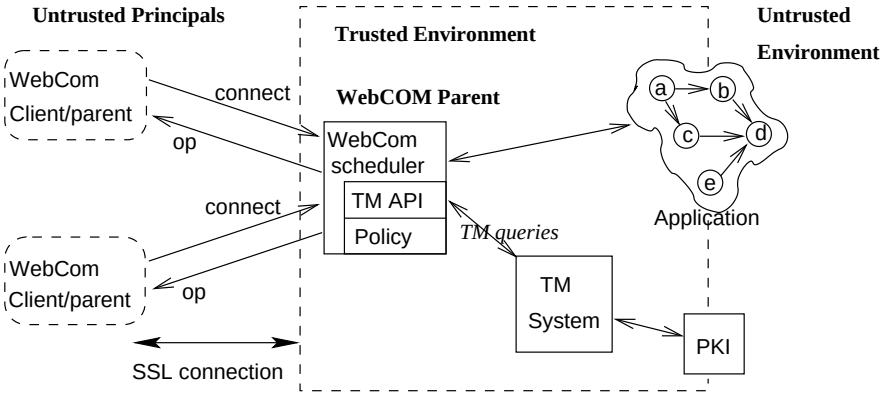

Figure 2: WebCOM-KeyNote Architecture

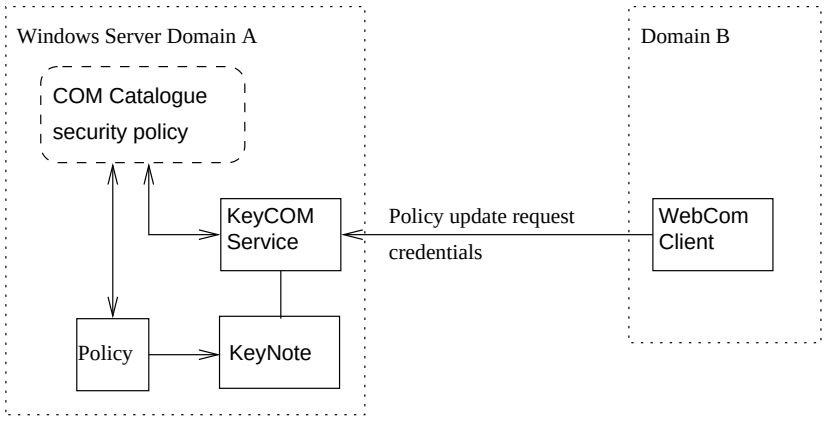

Figure 3: COM+ Authorisation Service

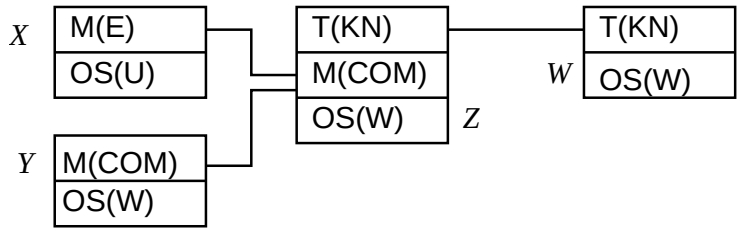

Figure 4: Interoperating Security Policies

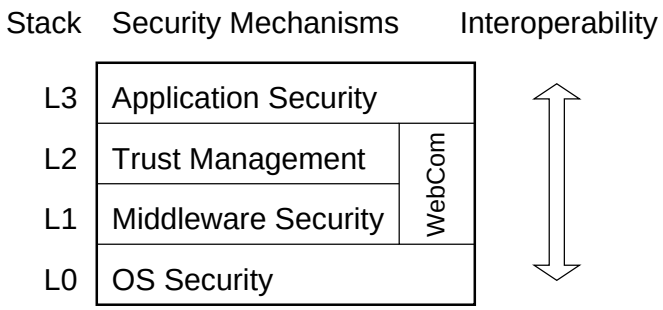

Figure 5: Stacked Security Architecture in WebCom

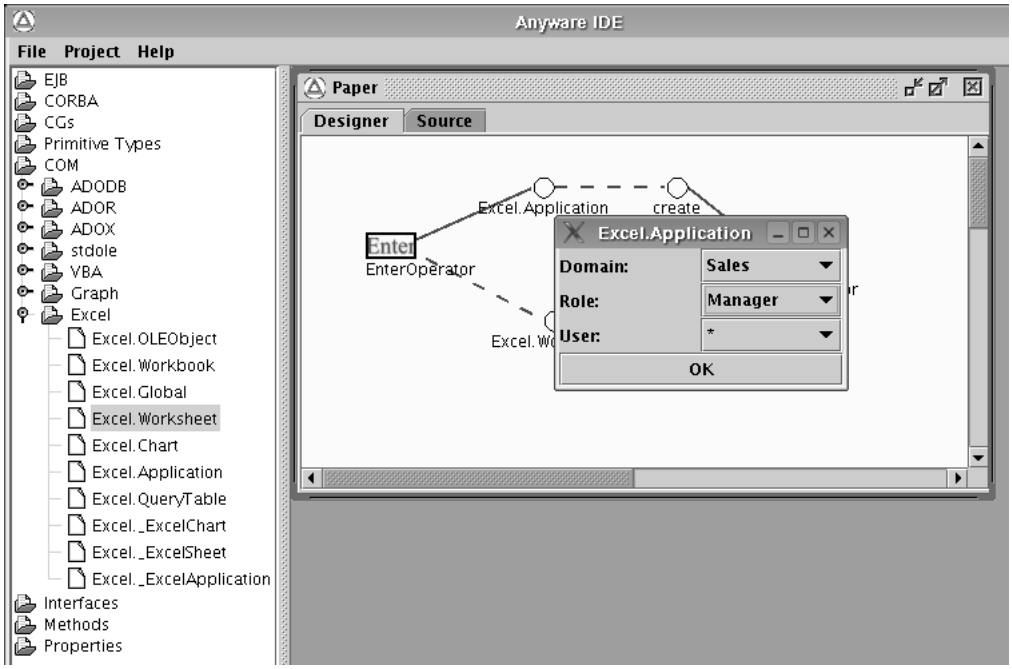

Figure 6: The WebCom Integrated Development Environment