Dynamic Administrative Coalitions with WebCom$_{DAC}$

Simon N. Foley
Department of Computer Science
University College Cork
Ireland
s.foley@cs.ucc.ie

Barry P. Mulcahy
Department of Computer Science
University College Cork
Ireland
b.mulcahy@cs.ucc.ie

Thomas B. Quillinan
Department of Computer Science
University College Cork
Ireland
t.quillllinan@cs.ucc.ie

Abstract

User authorisation is traditionally coordinated by system administrators processing delegation requests. This overhead influences an enterprise's ability to respond efficiently and contributes to the total cost of ownership. We propose to reduce this bottleneck by supporting Dynamic Administrative Coalitions of ordinary users that hold administrative powers reflecting their authority. These coalitions support delegation and on-the-fly creation of new administrative coalitions constrained by business workflow rules. This paper describes a distributed system that integrates security and workflow to manage interoperable middleware security policies; providing a decentralised security administration architecture that is survivable against network and system failures.

Keywords: Distributed Security, Workflow, Administration, Collaboration Technologies, Trust Management.

1. Introduction

Conventional middleware security architectures such as CORBA [7], Enterprise Java Beans (EJB) [19] and Microsoft COM+/.NET [12] are based on Role Based Access Controls (RBAC) [10, 18]. 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.

Existing policy management tools such as IBM's Tivoli [8, 9] and Microsoft's Active Directory [2] provide a centralised point for policy configuration and maintenance over dispersed resources. These tools often incorporate a framework for specifying the conditions under which authorisations may be used. Common conditions on the use of authorisations include workflow rules (the order in which actions must be done) and temporal rules (when can an action take place). This collection of rules is commonly referred to as *business rules* as they reflect the regulations and organisational structure of the business.

These business rules are usually co-located with the global security policy in a centralised location, leading to problems such as a single point of failure and network bottlenecks. These problems can be mitigated by replicating the business rules and security policy at strategic locations in the enterprise. We suggest that an ideal structure for authorisation control should be completely distributed, allowing the policy to be spread out across the organisation, residing where it is needed.

Authorisation certificates provide a partial solution by helping to decentralise the authorisation policy. Also referred to as cryptographic credentials, they support the delegation of authorisation between public keys in trust management schemes [11, 17]. Controlling the conditions under which delegation may be performed is a difficult problem. For example, Tivoli uses a centralised service [8, 9] to ensure the business rules are upheld when delegations take place. This provides centralised control over a decentralised security policy.

We are interested in using certificate based schemes to provide decentralised policy coordination of existing centralised security mechanisms. In [4], a Microsoft COM+ service is described that updates the underlying COM+ RBAC security policy according to user requests. An (ordinary) user provides, as part of her request, authorisation certificates which prove that she is entitled to hold the appropriate COM+ authorisation. Users delegate authorisation (held) for components to others by signing appropriate certificates. Delegation can be done without having to liaise with an authorisation server: every user effectively becomes an administrator for those components over which they have authority. This reduces system administrator workload while shifting authority to those most familiar with the application component and its security requirements: the application users.

While this approach pushes administration authority into the user space it does not support the enforcement of business rules. Once granted authorisation, there are no controls on when or how a user may delegate held authorisation to others: the user may act contrary to the proper procedures in the enterprise. This paper proposes a distributed architecture for coordinating business rules.

Enforcing business rules in a decentralised security policy environment requires coordination of distributed administrative actions. WebCom [3, 15] is a distributed secure and fault-tolerant architecture that can be used to coordinate the distributed execution of application components across a network. WebCom applications are developed as Condensed Graphs [14] of middleware components, and these are executed on their corresponding architectures as coordinated by WebCom.

We have argued [6] that WebCom and its underlying theory--Condensed Graphs [14]--is particularly effective at securely coordinating application workflow. In this paper we describe how WebCom [3] can be used to effectively coordinate delegation of authorisation based on business rules, providing for Dynamic Administrative Coalitions (DACs). The result is a distributed administration system that is fault-tolerant, coordinates decentralised security policies and pushes administration into the user space.

The paper is organised as follows. Section 2 introduces Condensed Graphs as a basis for distributed administrative workflows. Section 3 describes the basic architecture of the WebCom metacomputer and how it is used to control the execution of Condensed Graphs across a network. Section 4 describes a high-level workflow model for building DACs and how they are in turn encoded as Condensed Graphs. Section 5 outlines an implementation of the model (WebCom DAC) applied to middleware RBAC policies.

2. Condensed Graphs

In this section we outline Condensed Graphs (CG) and demonstrate how we can use them to encode workflow rules that include administrative actions. Like classical dataflow [1], the Condensed Graphs model [14] is graph-based and uses the flow of entities on arcs to trigger execution. Condensed Graphs are directed acyclic graphs in which every node contains not only operand ports, but also an operator and a destination port. Condensed Graphs provide an excellent framework for specifying and executing distributed workflow.

2.1 Application Workflow

A Condensed Graph is a collection of nodes where the connecting arcs represent the flow of execution. Node operators map to machine primitives, such as simple arithmetic operations (for example, + , -, ≤); nodes can also encapsulate more complex operations that correspond to the invocation of software components (on possibly different machines). Examples include COM components, EJB components, or even a mix of both. Condensed Graphs are so called because their nodes may be condensations, or abstractions, of other Condensed Graphs, this can be used to incorporate recursion in a workflow.

Example 1 Figure 1 gives a simple CG workflow for capturing a single business deal. The nodes *Price Deal, Analyse Risk, Buy* and *Sell* correspond to appropriate software components. The node *≤500* is an arithmetic function which determines whether the deal may be made without delay or if further checks should be made. The *ifel* nodes control the evaluation path of the workflow based on the value of the deal and the risk involved.

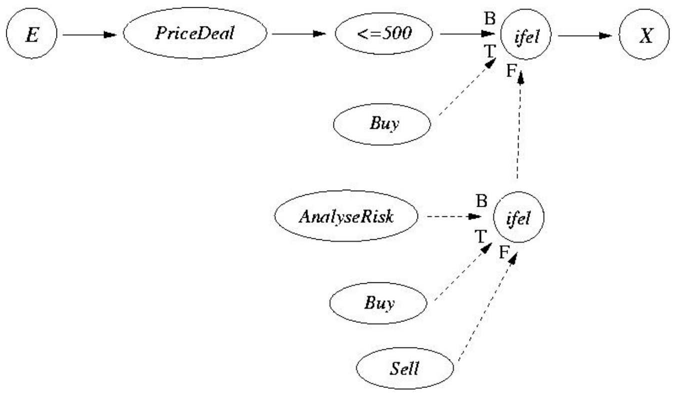

Figure 1: Application Workflow: Capture Deal

A node becomes executable when all its operands, operator and destinations are available. As one node is executed, its destination becomes the operand for the next connected node. In this way the execution order of the graph nodes determines the progression of the workflow. Condensed Graphs allow the combining of availability-driven computation and coercion-driven computation in a single graph. This is represented in the arcs connecting graph nodes. A solid arc indicates that the source node will be executed once it has its operands and operator (availability-driven). A dotted arc indicates that the source node will only be executed if required by the progression of the workflow (coercion-driven). Every graph has a single E and X node, which define the enter point and exit point respectively of the graph. The ifel (if-then-else) operator is specific to the CG language and provides a mechanism for specifying alternate flows depending on a boolean input. The 'then' and 'else' branches are resolved according to the boolean operand. A further explanation of the CG firing rules can be found in [14].

2.2 Administration Workflow

By encapsulating administrative actions in CG nodes we can coordinate decentralised security policies. This allows a *distributed* workflow for controlling administration, a capability lacking in other approaches, such as Tivoli [8, 9] and Active Directory [2].

Example 2 Figure 2 gives an example of the administrative actions required to appoint a Trading Manager for a company. When executed, the node *Select Manager* provides a user interface for specifying a unique identifier for the appointee, such as a userid. This information is passed to the two nodes *GrantPerm:Analyse Risk* and *GrantPerm:Price Deal*, which may execute in parallel (as they are not dependent on each other). The *GrantPerm* nodes are then responsible for granting the specified permissions to the appointee (for example, update a COM+ catalog with the permissions required to perform a *Price Deal* action and update an EJB container with the permissions required to perform an *Analyse Risk* action).

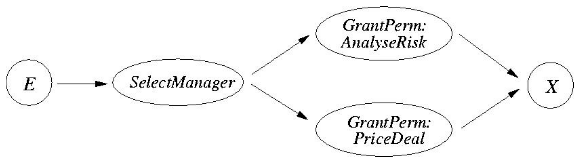

Figure 2: Administration Workflow: Trading Manager

3. WebCom

WebCom [3, 15] is a distributed, secure and fault-tolerant architecture that can be used to coordinate the distributed execution of Condensed Graph application components across a network. The WebCom platform is currently implemented as a Java application, which can communicate with other WebCom Coordinators to coordinate the execution of graphs (on a peer-to-peer or a master-client basis). WebCom frees an application writer from the burden of coding fault-tolerance, load-balancing and security requirements within the application.

3.1 Supporting Distributed Workflow in WebCom

WebCom applications are developed as XML coded Condensed Graphs [13, 14] of software components, and these are executed on their corresponding architectures as coordinated by WebCom. Taking the simple workflow example from Section 2.1 we can see in Figure 3 how WebCom might coordinate the different nodes of the graph among participating WebComs. The decision on where to schedule graph nodes for execution is based on the compatibility of the host platform with the software components, load-balancing, fault-tolerance, and security. WebCom security [3] is based on trust management and/or the security mechanisms of the underlying systems [4].

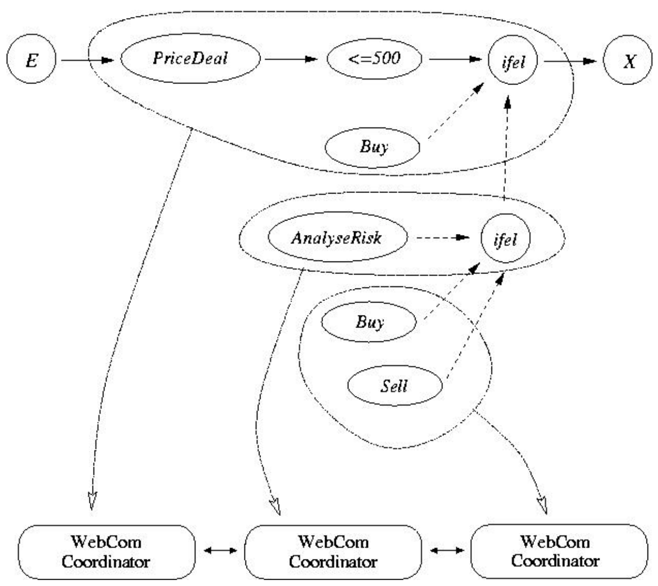

Figure 3: WebCom Coordination of Graph in Figure 1

3.2 Supporting Distributed Administrative Workflow in WebCom

In our model, administrative actions are incorporated into the workflow rules at a graph level. This provides a distributed mechanism for controlling security policies across existing (legacy) systems, for example, middleware servers such as Sun's EJB server. However it exposes the requirement for a security bridge between the graphs granting authorisations (Section 2.2) and the legacy systems. This is dealt with using Authorisation Servers.

Authorisation Servers
The need for bridging between graphs and the authorisation policy on legacy systems is provided by Authorisation Servers. The Authorisation Servers implement the required bridge by interpreting the authorisations granted at the graph level and updating the legacy policy configuration. How we determine the system specific permissions required to perform a task is covered in Section 4.2

The distributed administration workflow in Section 2.2 is supported by Authorisation Servers as follows. The *SelectManager* node is scheduled by WebCom to a suitable domain so that an authorised user can appoint a Trading Manager. The appointee's information (for example, a public key) is then passed to the *GrantPerm* nodes which are executed by WebCom and produce the appropriate authorisation certificates (credentials). WebCom can then send these credentials directly to the appropriate Authorisation Server(s). The credentials can also be passed to the appointee, providing a decentralised security policy. The appointee can then intiate the request (offline) to the Authorisation Servers to update the existing policy (for example, EJB container permissions).

4. Dynamic Administrative Coalitions (DACs)
In the previous sections we demonstrated the distributed control of existing (centralised) security policies using Condensed Graphs. The examples used have been simple graphs, where nodes correspond directly to single administrative actions (action-centric model). This provides a fine-grained control over security administration. However, for a large or complex set of rules the corresponding graphs become increasingly difficult for a developer to write.

We propose structuring the description of business rules as a set of activities (activity-centric model) that are translated into a Condensed Graph implementation. Specifying the business rules as a collection of individual activities is intuitive for a developer and provides an abstraction of the organisation's structure. We refer to the distributed control of authorisation using an activity-centric model as a Dynamic Administrative Coalition (DAC).

4.1 Activity Sets
We define an activity according to the definition in [5] for a Computer Supported Collaborative Working (CSCW) system. An individual activity has the following components:

- **join,leave:** These define the sets of possible actions that a user may engage to join and leave the activity, respectively.
- **do:** Once a user is a member of the activity, she participates in the activity by engaging actions from the set **do**.
- **start,finish:** Users may participate in an activity only during the activity's lifetime. The activity must start with an action from **start**. An action from **finish** is necessary to terminate the activity.
- **conclude:** Defines the sequences of activity actions, that may lead to a successful conclusion of the activity.

This characterisation forms the basis for specifying the operational requirements of an activity. In this paper we consider an activity to have an implicit **start** and **finish**, that is, an activity is started if it has participants, and stopped when all participants have left. Support for explicit **start**, **finish** and **conclude** events is managed using Condensed Graph concurrency primitives and is not described in this paper.

Example 3 An activity for a Trading Manager is defined in Figure 4. In this activity a Trading Manager is appointed through an `AppointTradingManager` action. Once appointed he may leave the activity by a `Resign` action. A participant of this activity can perform the tasks `Analyse Risk`, `Price Deal`, `Appoint Clerk` and `Resign`.

Activity:	`Trading Manager`
join:	`{Appoint Trading Manager}`
leave:	`{Resign}`
do:	`{Analyse Risk, Price Deal, Appoint Clerk, Resign}`

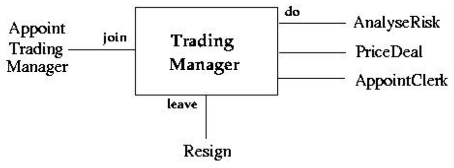

Figure 4: Activity Specification: Trading Manager

The activity specification for a Trading Manager can be encoded as a Condensed Graph, as depicted in Figure 5. In our implementation, activity specifications are coded in XML and automatically translated to Condensed Graph XML [13] using an XML translator.

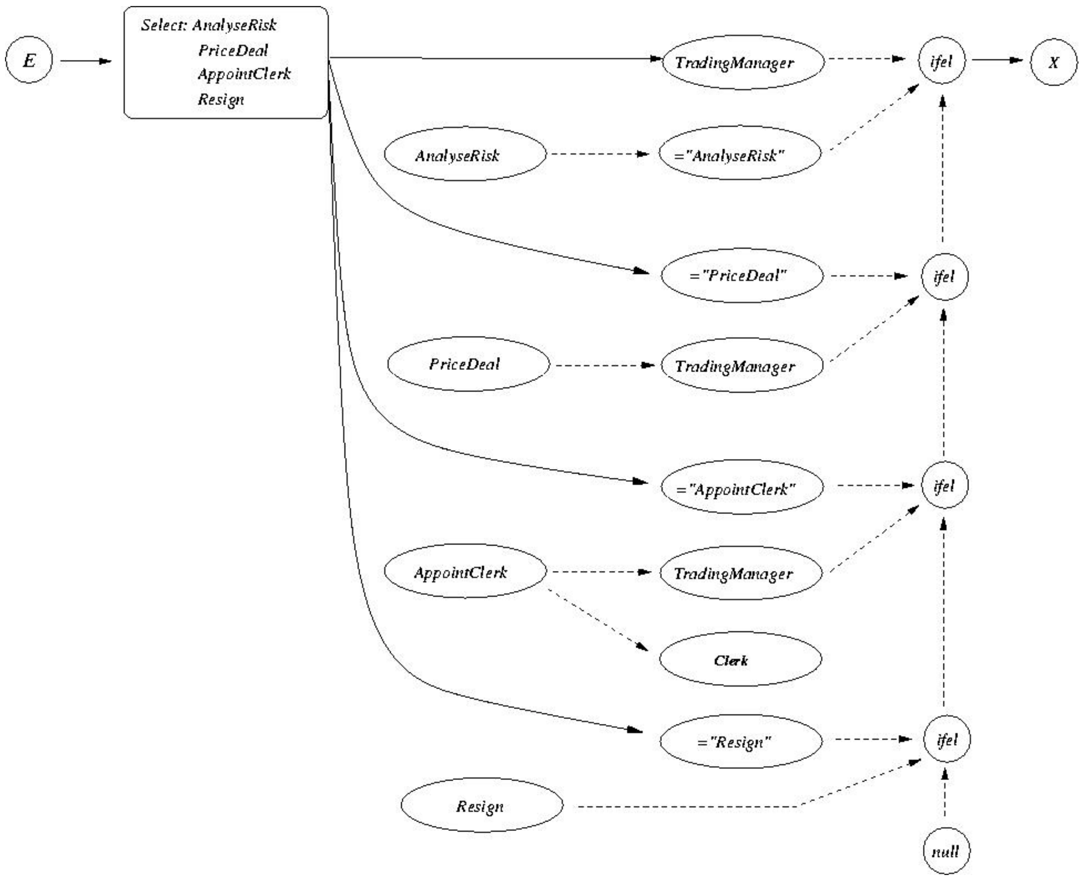

Figure 5: Condensed Graph: Trading Manager

4.2 Action Authorisation

The semantics for an activity (Section 4.1), define the workflow rules of that activity. However, we have not yet considered the authorisation requirements to execute their actions on the underlying systems. For each activity, there are a set of actions that activity members can engage in (**do** actions). In our model, when a principal becomes a member of an activity they are granted the appropriate permissions required to perform the specified actions (in that activity).

The authorisation requirements to perform an action are obtained from the underlying security policies (for example, COM and EJB) by the Authorisation Servers. If the **do** set of an activity consists of two actions: a, and b; then becoming a member of that activity corresponds to the granting of permissions that allow execution of these actions. That is, `Grant(Perm(a), Perm(b))`, where holding `Perm(a)` implies authorisation to perform action `a`. Section 5 considers how this is implemented in practice.

Where one activity may appoint members of another activity our model supports a form of constrained delegation. Activity participants are able to delegate a permission without holding it themselves. This is important as it ensures that an activity member never holds more authorisation than is necessary to perform their allotted tasks.

Example 4 Consider the activities for a Trading Manager (Figure 4) and a Clerk (Figure 6), where a Trading Manager can appoint a Clerk. In this case the Clerk holds a permission that the Trading Manager does not hold: `(Perm(Capture Deal))`.

Activity:	`Clerk`
join:	`{Appoint Clerk}`
leave:	`{Resign}`
do:	`{Capture Deal, Resign}`

Figure 6: Activity Specification: Clerk

To allow the appointment of a Clerk we define the Trading Manager's permission set as:

```
{Perm(Analyse Risk)
 Perm(Price Deal)
 Perm(Grant(Perm(Capture Deal)))
 Perm(Resign)}
```

The Trading Manager does not hold `Perm(CaptureDeal)`, but she can grant the permission to a Clerk. This upholds the business rules by denying the Trading Manager unnecessary authorisation. In the next section we detail the mechanisms which ensure that a Trading Manager cannot appoint herself as a Clerk.

5. WebCom DAC

In this section we use the proposed framework to coordinate authorisation in middleware RBAC policies. The business rules for an enterprise are developed as a collection of interacting activities; this policy is translated into the corresponding Condensed Graph implementation, which is in turn coordinated by a collection of WebCom Coordinators. These coordinators constrain the execution of the administrative actions according to the generated graph; the Authorisation Servers provide the bridge between the graphs and the underlying security policies. This demonstrates distributed control over existing centralised security policies using an activity-centric model. Figure 12 gives an overview of the entire process described in Sections 5.1 to 5.5.

5.1 DAC Policy Tool

The Policy Tool is an Integrated Development Environment (IDE) for constructing Dynamic Administrative Coalitions (DACs) using the activity-centric model described in Section 4. Activity actions are selected from a database of middleware components. Details of the database implementation can be found in Section 5.5.

A graphical representation of the business rules for a share trading company is depicted in Figure 7. Activities with shared actions are shown using connecting lines between the activities. The workflow requirements, administration actions and their associated rules are translated into the corresponding Condensed Graphs by the Policy Tool. The workflow is then ready for execution by WebCom.

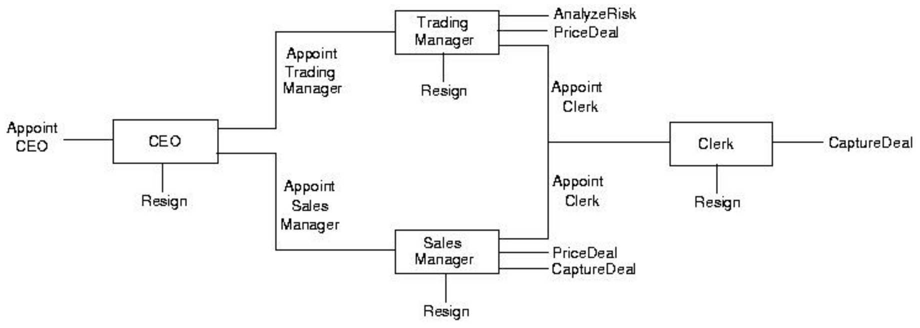

Figure 7: DAC for a Share Trading Company

5.2 DAC Coordinators - WebCom

WebCom schedules the nodes of the workflow graph to a suitably authorised domain, where a domain is identified by a public key. When an activity participant authenticates herself to WebCom, the workflow nodes, for which she is authorised, will be scheduled by the WebCom coordinators to her domain. This provides activity participants with a selection of their available (authorised) actions.

5.3 DAC Authorisation

Workflow nodes that encode administration actions require a mechanism to allow delegation of authorisation. Our implementation uses the Trust Management system KeyNote [11] which provides a simple credential notation for expressing both security policies and delegation. Authorisation is conferred in the form of digitally signed public key credentials that bind public keys to the authorisation to perform actions. Authorisation held in a credential can then be delegated to another public key by signing an appropriate cryptographic credential. The action of delegation is encoded as the special `Grant` graph node (defined in Figure 9).

However, we need to ensure that principals cannot bypass the enforcement mechanisms and violate the business rules. For example, preventing a Trading Manager delegating `Perm(Capture Deal)` to herself or anyone else who is not a Clerk, that is, bypassing the workflow rules for the activity.

```
KeyNote-Version: 2
Comment: Threshold Scheme
Local-Constants:
  KCEO = "rsa-hex:0fb34562a2cf230001"
  KWebCom = "rsa-hex:4ac34891bacddf2311"
Authorizer: "POLICY"
Licensees: KCEO && KwebCom
Conditions: App_Domain == "WebComDAC" &&
 (Domain=="alice.company.com container") &&
 ((Function=="AnalyseRisk") ||
  (Function=="PriceDeal") ||
  (Function=="CaptureDeal"));
Signature: ...
```

Figure 8: Policy for Threshold Scheme in KeyNote

The rules are upheld by ensuring that every `Grant` action is done with the authority of the principal involved in the action *and* the WebCom Coordinator. The Coordinator can then ensure that the workflow rules of the activity are upheld. This is implemented by treating WebCom Coordinators (identified by their own public key) as trusted third-parties, who must also counter-sign any delegation by a principal. This is enforced using a threshold policy, as seen in Figure 8. This may seem like a complex task, but the rules for delegation actions are automatically encoded in the graphs produced by the Policy Tool. The business rules can then be enforced transparently by the WebCom Coordinators.

Example 5 An example of the `Grant` operation is implemented by the graph in Figure 9. The `Grant` graph takes as input a collection of base conditions (*conds*). The activity participant is then prompted for additional information *GetAuthorisor*, *GetLicencee* and *GetConditions* (this allows additional user-specified conditions). The user credential can then be produced (by the node *UserDelegation*) and provided to the licensee (*OutputCredential*). If the specified licensee does not equal the authoriser, the second credential will be produced by the node *WebComDelegation*. This prevents self-delegation of authorisation. The second credential is provided to the licensee by the node *OutputCredential*. The Grant graph can be used to implement the *AppointClerk* node seen earlier in Figure 5.

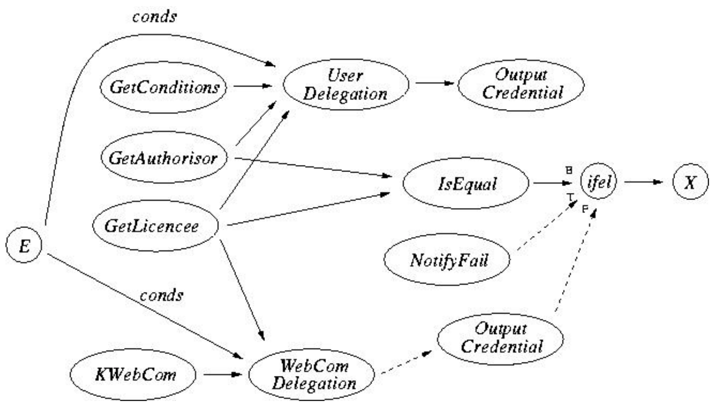

Figure 9: Condensed Graph: Grant

Credentials produced as part of the workflow are scheduled by WebCom to the licensee's domain, thereby decentralising the security policy and providing the relevant portions to those who require it. This is an on-demand administration model, licensees can send administrative requests to the authorisation servers as the need arises. The credentials can also be scheduled directly to the Authorisation Server(s) by WebCom, providing an availability-driven administration model. An example of the counter-signed credential for a Clerk can be seen in Figure 10.

```
KeyNote-Version: 2
Local-Constants:
  KCEO = "rsa-hex:0fb34562a2cf230001"
  KWebCom = "rsa-hex:4ac34891bacddf2311"
Authorizer: "KWebCom"
Licensees: KClerk
Conditions: App_Domain == "WebComDAC" &&
 (Domain=="alice.company.com container") &&
 ((Function=="CaptureDeal") &&
  (__action_authorizers=KClerk));
Signature: ...
```

Figure 10: WebCom signed credential for a Clerk

5.4 DAC Authorisation Servers

Gaining authorisation on the middleware services is achieved by presenting the relevant Authorisation Server with the credentials from the granting activity participant and the counter-signed credentials from WebCom. Given the Trust Management policy and the provided credentials, the Authorisation Server can determine whether a particular public key is authorised to request a particular policy change on the underlying middleware service. The requested policy change information is stored in the credential assertions using an extended RBAC format detailed in the next section. Implementations of the COM+ and EJB authorisation Servers are described in [4].

5.5 Middleware Interrogation

In order to specify business rules for existing middleware policies, we need to ascertain for each middleware service the available components and the system specific permissions required to execute each component. Currently, there are implementations of the interrogation agents for Microsoft's COM+ and Sun's EJB server (which complement the Authorisation Servers detailed in Section 5.4). The elicited information is stored in an extended RBAC format [4] which includes *Domain* and *ObjectType* information. A Domain is scoped by the service instance, for example, a COM+ machine or an EJB container. Roles are members of these domains. Permissions are defined in the context of service specific *ObjectTypes*, for example, COM+ permissions apply to COM objects whereas EJB permissions are method references applied to Java classes. This extended format can represent different middleware RBAC policies. An example of the Conditions field in a Clerk's credential (as written by a Trading Manager) can be seen in Figure 11, where `alice.company.com` is an EJB server and `TraderBean` is a bean deployed on the server. In this case, the Clerk has been granted permission to execute the bean method CaptureDeal .

```
Conditions:
 (Domain=="alice.company.com TraderBean") &&
 ((Function=="CaptureDeal") &&
  (Permission=="Grant"));
```

Figure 11: Conditions Field for a Clerk's Credential

6 Discussion and Conclusion

In [4] the KeyNote Trust Management system is used to provide interoperability support between COM+/.NET and Enterprise Java Beans middleware security policies. Middleware authorisation policies are encoded in terms of KeyNote cryptographic certificates, and vice-versa. This provides a unified view of security of a heterogeneous middleware application system, and also provides the basis for the support of centralised and decentralised RBAC middleware security. However, once authorisation has been delegated to a principal, there is little fine-grained control over how that held authorisation may be subsequently delegated to others.

In this paper we address this issue by introducing workflow business rules that are used to specify the procedural requirements for administration. These workflow rules are used to constrain the delegation actions. The WebCom metacomputer is used to provide an architecture for coordinating the execution of administration actions according to the specified workflow rules. Unlike existing security administration systems that rely on centralised mechanisms for (limited) business rules, complex workflows can be managed across a distributed collection of trusted WebCom coordinators. This provides for an on-demand administration environment.

1. Interrogation of middleware services
2. Database populated
3. Policy Tool initialised
4. Application developed
5. Application translated to graphs
6. Workflow distributed and executed
7. Administrative (delegation) actions performed
8. Authorisation Server notified of administration action
9. Middleware policy updated to reflect changes

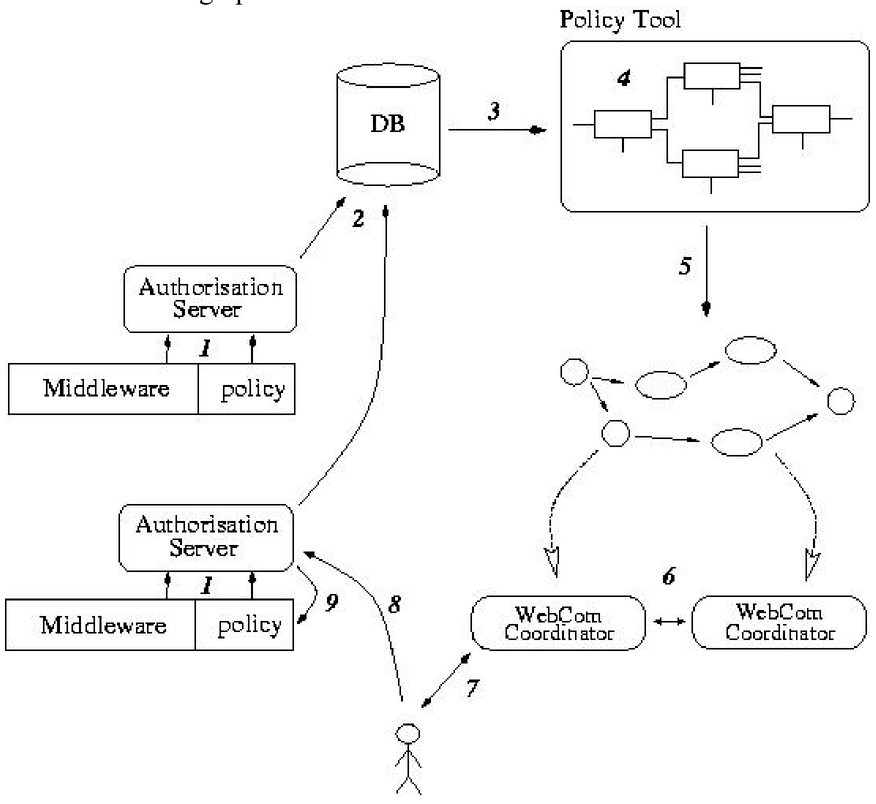

Figure 12: Overview of WebCom DAC

An alternative approach to constraining delegation is proposed in [16]: authorisation credentials encode regular expressions that denote the acceptable delegation `sequences' on any delegation path in which they occur. The computational complexity of searching for a valid authorisation path in this kind of delegation network is such that its application is limited in practice. The advantage of our more operational approach is that complex workflow rules involving a mix of administration and application actions can be encoded as a condensed graph.

Rather than specifying workflows in terms of graphs of administration actions (action-centric), we propose that administration workflow is structured in terms of coalitions of users collaborating on common activities (activity-centric). Complex workflows can be specified in a natural way in terms of concurrent activities synchronising on common actions. These activity specifications are in turn (automatically) translated into a condensed graph implementation whose execution is coordinated by WebCom. This allows the ad-hoc creation of virtual teams outside the normal scope of an organisations structure.

Acknowledgements

The support of the Informatics Research Initiative of Enterprise Ireland is gratefully acknowledged. This work was done in collaboration with the Centre for Unified Computing in UCC. Our thanks to John Morrison for his support and access to the WebCom infrastructure.

References

1. Arvind and Kim P. Gostelow. “A Computer Capable of Exchanging Processors for Time”. *Information Processing 77 Proceedings of IFIP Congress 77*, Pages 849-853, Toronto, Canada, August 1977.
2. Microsoft Corporation. Microsoft active directory technical white paper. Technical report, 2000-2004. Microsoft TechNet.
3. 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.
4. 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.
5. S.N. Foley and Jacob J.L. “Specifying Security for Computer Supported Collaborative Working”. *Journal of Computer Security*, 3(4):233-254, 1994/1995.
6. S.N. Foley and J.P Morrison. “Computational Paradigms and Protection”. In *ACM New Computer Security Paradigms*, Cloudcroft, NM, USA, 2001. ACM Press.
7. The Object Management Group. Common object request broker architecture (corba/iiop). Technical report. http://www.omg.org/technology/documents/formal/corba iiop.htm.
8. Gunter Karjoth. “The Authorization Model of Tivoli Policy Director”. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC 2001)*, pages 319-328. IEEE Computer Society, December 2001.
9. Gunter Karjoth. “Access Control with IBM Tivoli Access Manager”. *ACM Transactions on Information and System Security (TISSEC)*, 6(2):232- 257, 2003.
10. David F. Ferraiolo D. Richard Kuhn and Ramaswamy Chandramouli. *Role-Based Access Control.* Artech House, 2003.
11. J. Ioannidis M Blaze, J. Feigenbaum. The KeyNote Trust-Management System Version 2. September 1999. Internet Request For Comments 2704.
12. Microsoft Corporation. Microsoft Platform SDK. The Component Object Model. Microsoft Developer Network., 0.9 edition, October 1995.
13. John P. Morrison and Philip D. Healy. “Implementing the WebCom 2 Distributed Computing Platform with XML”. In *IEEE Proceeding of the International Symposium on Parallel and Distributed Computing*, 2002.
14. J.P. Morrison. “Condensed Graphs: Unifying Availability-Driven, Coercion-Driven and Control-Driven Computing”. PhD thesis, Eindhoven, 1996.
15. 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.
16. Babak Sadighi Firozabadi Olav Bandmann, Mads Dam. “Constrained Delegation”. In *IEEE Symposium on Security and Privacy*, May 2002.
17. R Rivest and B Lampson. “SDSI - A Simple Distributed Security Infrastructure”. In *DIMACS Workshop on Trust Management in Networks*, 1996.
18. Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. “Role-Based Access Control Models”. *IEEE Computer*, 29(2):38- 47, 1996.
19. Sun Microsystems. Enterprise JavaBeans(tm) Specification, Version 2.1, June 2003. http://java.sun.com/products/ejb/docs.html.