RBACManager: IMPLEMENTING A MINIMAL ROLE BASED ACCESS CONTROL SCHEME ($RBAC_M$) UNDER THE WINDOWS NT 4.0 WORKSTATION® OPERATING SYSTEM

W. Caelli[1], A. Rhodes[2]

Abstract

RBACManager is a Microsoft "Windows NT" system that has been developed to administer security using a minimal, role-based access control ($RBAC_M$) model. A demonstration environment is outlined, prior to a two-phase demonstration/tutorial. The demonstration emphasizes the security administration aspects of $RBAC_M$ and then details the results of an execution of a prototype application. Technical aspects of the implementation are presented to provide an insight into the mapping of roles into Windows NT "groups". Future research, particularly for workflow environments, is discussed.

Introduction

This paper is the second of two related papers describing the design and implementation of a minimal Role Based Access Control ($RBAC_M$) framework to run on top of the Windows NT 4.0 Workstation Operating System. These papers are intended to provide a solid foundation for future investigation into higher level RBAC models that are "active", rather than "passive", in nature. The associated paper is entitled "The Design of a Minimal Role Based Access Control System under the Windows NT 4.0 Workstation® Operating System" and has been submitted for publication.

RBACManager is a Windows NT application that has been developed to administer system security using a role based access control ($RBAC_M$) model. This allows security administration to be centrally managed at a higher abstraction level, which leads to simpler organisational security implementation and therefore fewer errors. Unlike recent applications that have focused on integrating RBAC at the application level, *RBACManager* integrates the RBAC framework at the operating system level. This provides facilities that are sufficiently flexible to support a wide range of applications with minimal customization.

This paper is intended to demonstrate the use and application of *RBACManager*. The sections within the paper are structured as follows:

- **$RBAC_M$ Implementation Details**
  Outlines briefly the software technologies to implement $RBAC_M$.
- **$RBAC_M$ Demonstration Details and Entities**
  Describes the conditions and scope of the demonstration. Presents the entities and relationships used throughout the demonstration
- **$RBAC_M$ Demonstration**
  Demonstrates the role based nature of RBAC Manager and provides evidence that RBAC Manager successfully manages security at a high level. This section uses a step-by-step tutorial approach to illustrate the application and use of RBAC Manager. In particular, as RBACManager manages the underlying Windows NT security mechanisms, this section highlights the impact on the Windows NT security entities as a result of an action taken in RBACManager.

[1] School of Data Communications, Queensland University of Technology, GPO Box 2434, Brisbane 4001, Australia.
[2] School of Computing Science, Queensland University of Technology, GPO Box 2434, Brisbane 4001, Australia.

- **RBAC$_M$ Summary**

  Discusses briefly some issues arising from the implementation of RBAC$_M$

- **Technical Highlights**

  Outlines many of the Windows NT security specifics that were required to successfully to implement RBAC$_M$

- **Future Research**

  Where does RBAC$_M$ go from here?

RBAC$_M$ Implementation Details

The implementation language chosen for RBAC$_M$ was the Python scripting language. Python is a high-level object-oriented programming language. As with other scripting languages, like Perl, Python is a dynamically typed language. The main RBACManager compiled Python program is 15,139 bytes in length.

Python provides a rich set of libraries that may be used. In addtion, PythonWin contains a Microsoft Foundation Class (MFC) based library with a rich interface to MFC, which was used extensively in the implementation of *RBACManager.* Python also allows module extensions to be created (using a language such as C) to implement features not found in standard Python. For RBAC$_M$ a module extension was created to provide an interface to the LAN Manager API and the Win32 Security API.

All aspects of Python are object-oriented. Python implements late binding of objects so the value of an object is resolved at run time through a dynamic name search. This feature was exploited by this implementation in that the same method name is applied to Role, User and FilePermissions classes.

RBAC$_M$ Demonstration Details and Entities

The exposition of RBAC$_M$ is conducted in two phases. Firstly, the security administration provided by *RBAC Manager* is outlined. This demonstrates the concept of RBAC$_M$ and how *RBAC Manager* fulfills the requirements of an RBAC$_M$ framework. It will also show that RBAC$_M$ simplifies security administration by providing a high level, centralized mechanism to administer security.

The second phase will involve demonstrating and providing evidence that *RBAC Manager* has enforced the security administered in phase 1. A minimal prototype application was developed to assist with phase 2. This shows that *RBACManager* provides a mechanism to successfully administer security while fulfilling the requirements of RBAC$_M$.

A number of entities and entity relationships are applied throughout this demonstration. These are presented below.

User Entities

User Name	Full Name	Password
Allan	Allan Miller	Allan
Brett	Brett French	Brett
Carolyn	Carolyn Landers	Carolyn
David	David Rogers	David
Ella	Ella Smith	Ella
Fran	Fran Urkhart	Fran
Geoff	Geoff Daken	Geoff
Helen	Helen Willis	Helen
Julie	Julie Handcock	Julie
Marsha	Marsha Yang	Marsha
Nathan	Nathan Ford	Nathan
Steve	Steve Soberon	Steve
Trent	Trent Bridge	Trent
Will	Will Dodds	Will
Yang	Yang Hilltop	Yang

Role Entities

Role Name	Role Description	Max. Users
Accounts Payable	Accounts Payable Role	2
Accounts Receivable	Accounts Receivable Role	2
Administration	Administration Role	3
Admittance	Admittance Role	2
Doctor	Doctor Role	2
Intern	Intern Role	1
Nurse	Nurse Role	4
Nurse Assistant	Nurse Assistant Role	2
Specialist	Specialist Role	1

Entity Relationships

Role-Role Membership (member roles that will form the role hierarchies)

Role Name	Member Roles
Accounts Payable	*Nil*
Accounts Receivable	*Nil*
Administration	*Nil*
Admittance	*Nil*
Doctor	**Intern** **Nurse**
Intern	**Nurse**
Nurse	**Nurse Assistant**
Nurse Assistant	*Nil*
Specialist	**Nurse**

Role-User Membership (valid users for a role)

Role Name	Member Users
Accounts Payable	▪ Marsha ▪ Nathan
Accounts Receivable	▪ Allan ▪ Geoff
Administration	▪ **Carolyn** ▪ **Helen** ▪ Steve
Admittance	▪ **Helen** ▪ Yang
Doctor	▪ **Brett** ▪ Ella
Intern	▪ Will
Nurse	▪ David ▪ Julie ▪ Trent
Nurse Assistant	▪ **Carolyn** ▪ Fran
Specialist	▪ **Brett**

Note that 1) Carolyn is in “Administration” and “Nurse Assistant”, 2) Helen is in “Administration” and “Admittance” and 3) Brett is in “Doctor” and “Specialist”.

Role Mutex (roles that will be mutually exclusive)

Role Name	Member Roles
Accounts Payable	Accounts Receivable
Accounts Receivable	Accounts Payable
Administration	*Nil*
Admittance	*Nil*
Doctor	*Nil*
Intern	*Nil*
Nurse	*Nil*
Nurse Assistant	*Nil*
Specialist	*Nil*

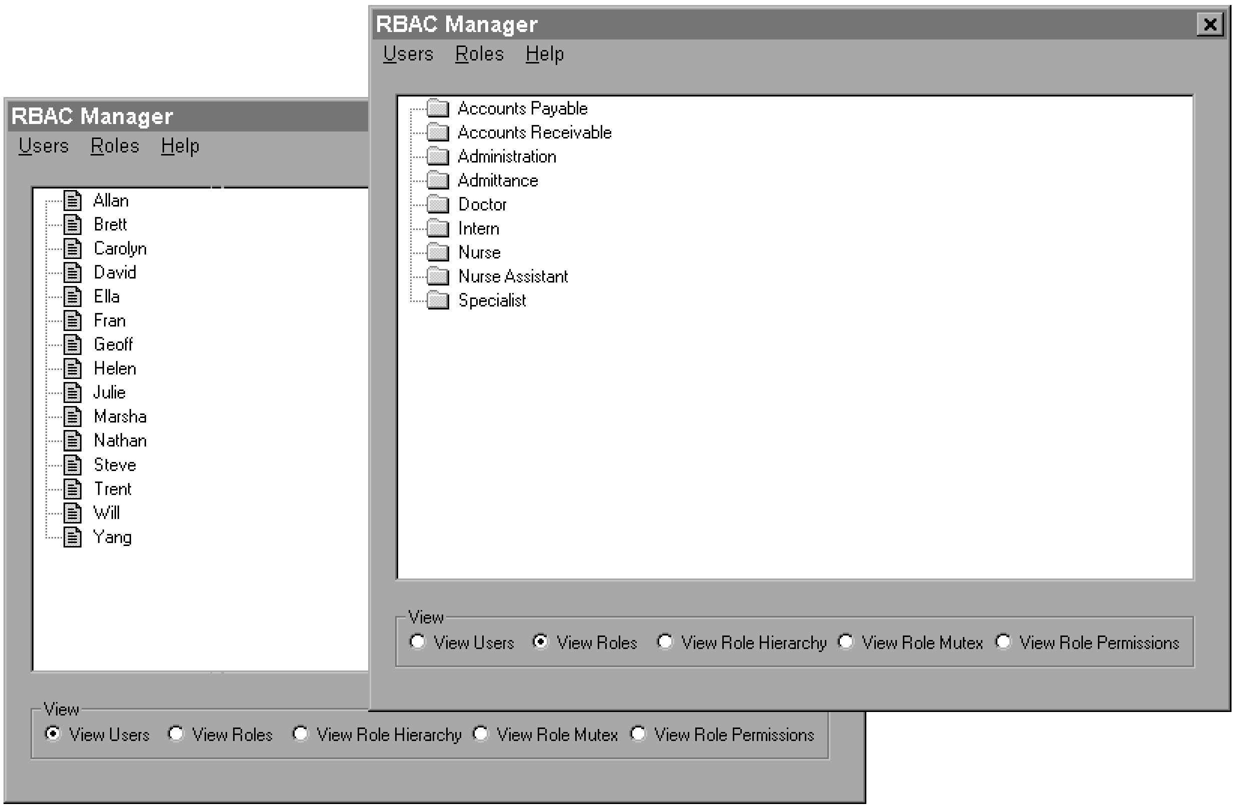
Role Permissions (permissions assigned to each role)

Role Name	Member File	Permissions
Accounts Payable	\RBAC Project\RBACDemo\payable	RW
Accounts Receivable	\RBAC Project\RBACDemo\receivable	RWX
Administration	\RBAC Project\RBACDemo\patient	R
Admittance	(* no file accessed *)	
Doctor	\RBAC Project\RBACDemo\treatment	X
Intern	\RBAC Project\RBACDemo\treatment	W
Nurse	\RBAC Project\RBACDemo\patient \RBAC Project\RBACDemo\treatment	W R
Nurse Assistant	\RBAC Project\RBACDemo\patient	R
Specialist	(* no file accessed *)	

RBAC$_M$ Demonstration – Phase 1: Security Administration

Adding Users and Roles

After adding all the users and roles presented in the previous tables, the *RBACManager User View* and *Role View* resemble the following two screens, respectively.



RBAC Manager

Users Roles Help

- Allan
- Brett
- Carolyn
- David
- Ella
- Fran
- Geoff
- Helen
- Julie
- Marsha
- Nathan
- Steve
- Trent
- Will
- Yang

View: [✓ View Users] View Roles View Role Hierarchy View Role Mutex View Role Permissions

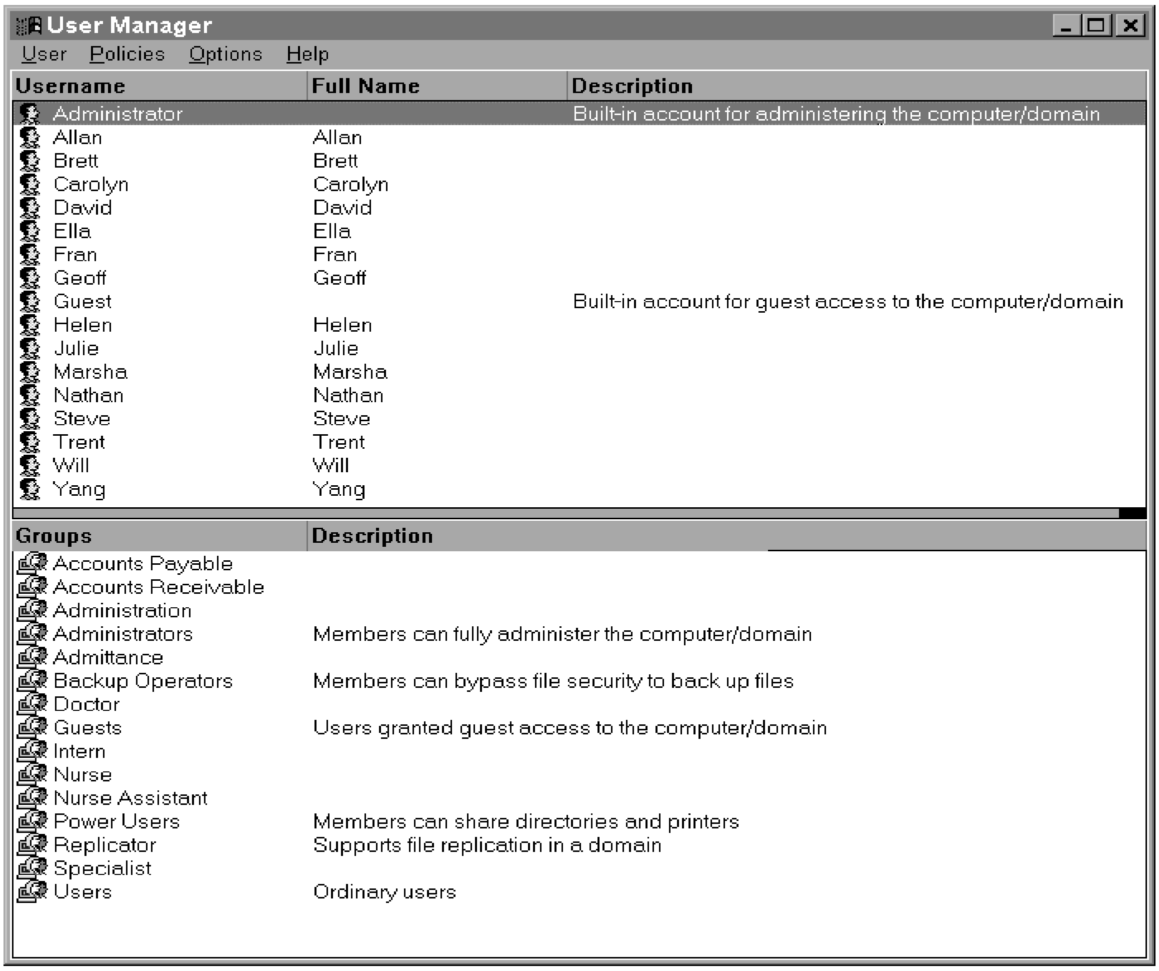
RBAC Manager

Users Roles Help

- Accounts Payable
- Accounts Receivable
- Administration
- Admittance
- Doctor
- Intern
- Nurse
- Nurse Assistant
- Specialist

View: View Users [✓ View Roles] View Role Hierarchy View Role Mutex View Role Permissions

Examination of the Windows NT Security Database using “User Manager” shows that the “users” and “roles” have been added. **Note that the roles have been added as groups**.

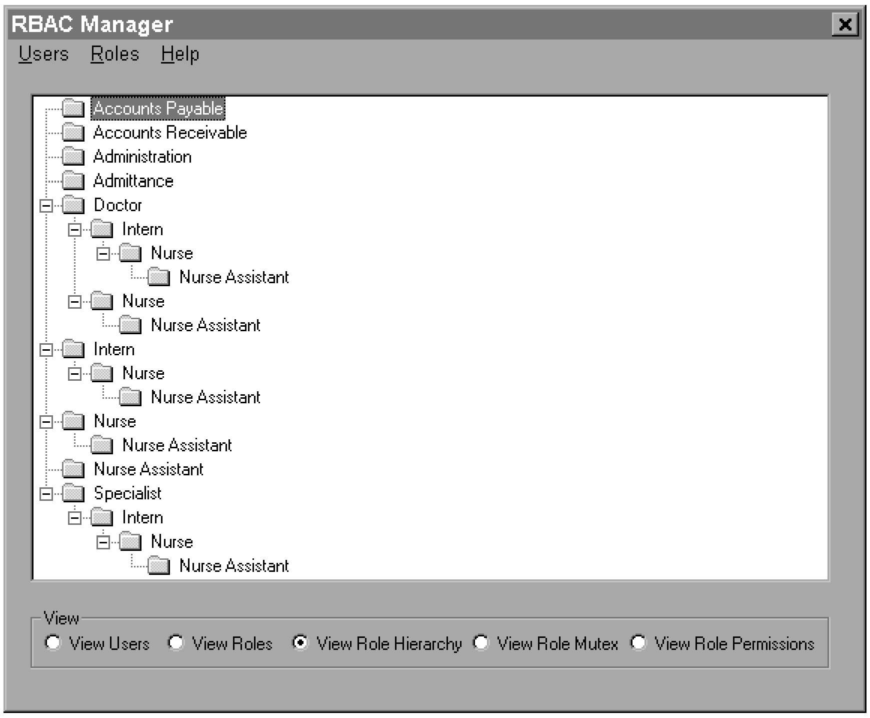


Username	Full Name	Description
Administrator		Built-in account for administering the computer/domain
Allan	Allan	
Brett	Brett	
Carolyn	Carolyn	
David	David	
Ella	Ella	
Fran	Fran	
Geoff	Geoff	
Guest		Built-in account for guest access to the computer/domain
Helen	Helen	
Julie	Julie	
Marsha	Marsha	
Nathan	Nathan	
Steve	Steve	
Trent	Trent	
Will	Will	
Yang	Yang	

Groups	Description
Accounts Payable	
Accounts Receivable	
Administration	
Administrators	Members can fully administer the computer/domain
Admittance	
Backup Operators	Members can bypass file security to back up files
Doctor	
Guests	Users granted guest access to the computer/domain
Intern	
Nurse	
Nurse Assistant	
Power Users	Members can share directories and printers
Replicator	Supports file replication in a domain
Specialist	
Users	Ordinary users

Assigning Actual Roles to “Roles”

After assigning the member roles, as presented in the *Role – Role Membership* table, the *RBACManager* role hierarchy view resembles:



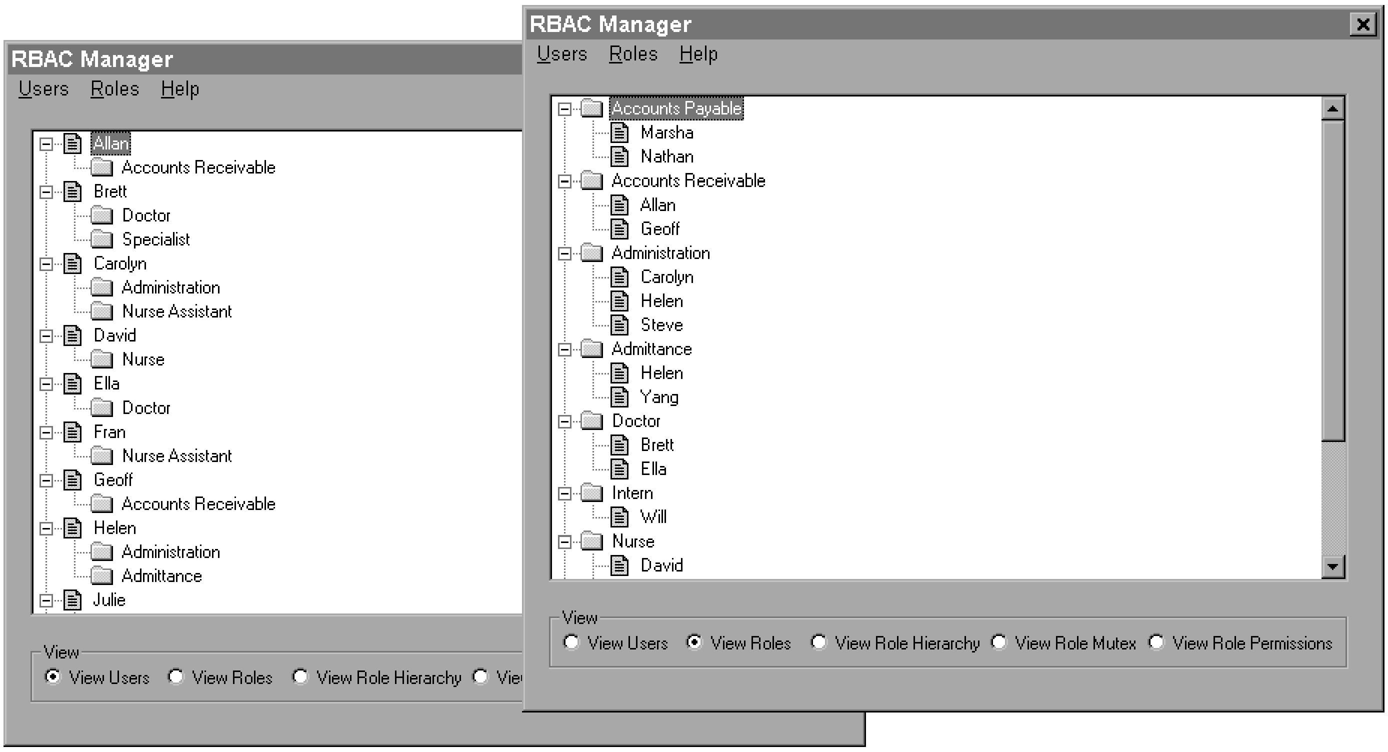
```
Accounts Payable
Accounts Receivable
Administration
Admittance
Doctor
    Intern
        Nurse
            Nurse Assistant
    Nurse
        Nurse Assistant
Intern
    Nurse
        Nurse Assistant
Nurse
    Nurse Assistant
Nurse Assistant
Specialist
    Intern
        Nurse
            Nurse Assistant
```

View: View Users View Roles [✓ View Role Hierarchy] View Role Mutex View Role Permissions

As **Windows NT doesn't allow local groups to be members of other local groups** this function does not actually modify any underlying structures in the Windows NT security sub-system. *RBACManager* separately controls the role hierarchy in its own database. This will be further examined in the RBAC$_M$ summary section.

Assigning Users to Roles

After assigning the users to the roles as specified in the *Role-User Membership* table the *RBACManager User View* and *Role View* resemble the following two screens respectively.



RBAC Manager

Users Roles Help

- Allan
  - Accounts Receivable
- Brett
  - Doctor
  - Specialist
- Carolyn
  - Administration
  - Nurse Assistant
- David
  - Nurse
- Ella
  - Doctor
- Fran
  - Nurse Assistant
- Geoff
  - Accounts Receivable
- Helen
  - Administration
  - Admittance
- Julie

View: [✓ View Users] View Roles View Role Hierarchy Vie

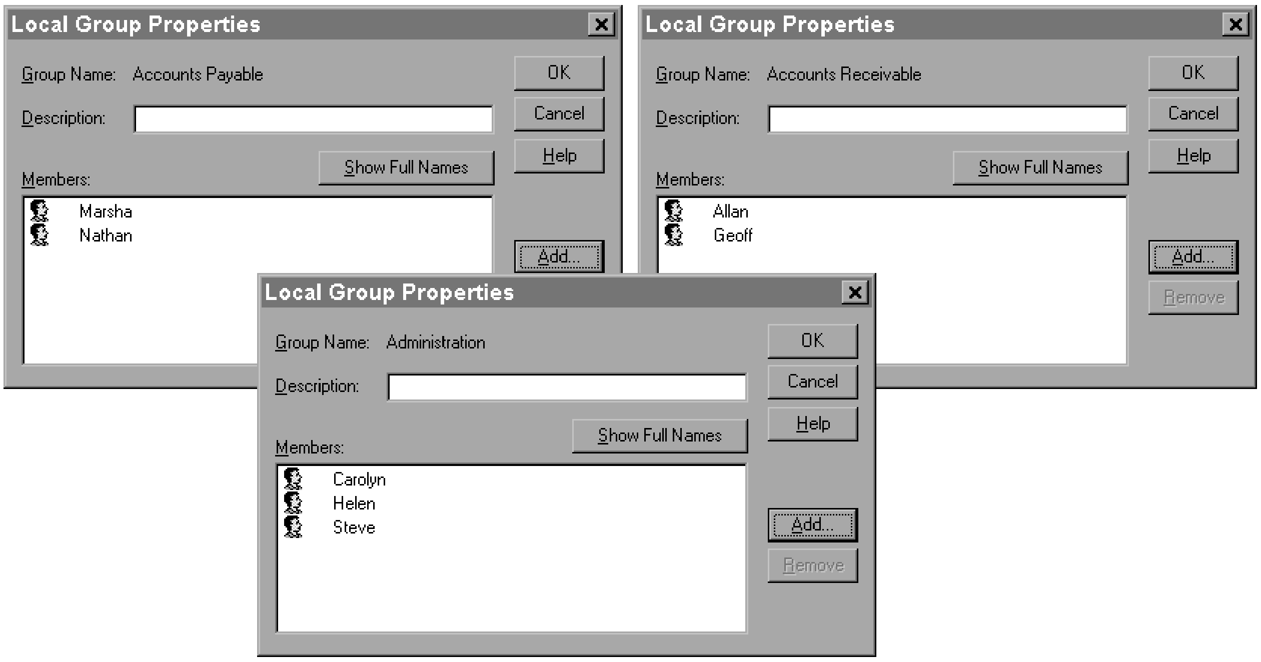
RBAC Manager

Users Roles Help

- Accounts Payable
  - Marsha
  - Nathan
- Accounts Receivable
  - Allan
  - Geoff
- Administration
  - Carolyn
  - Helen
  - Steve
- Admittance
  - Helen
  - Yang
- Doctor
  - Brett
  - Ella
- Intern
  - Will
- Nurse
  - David

View: View Users [✓ View Roles] View Role Hierarchy View Role Mutex View Role Permissions

When a user is assigned to a role, *RBACManager* updates the underlying Windows NT Security Database. Role members become members of Windows NT groups. Using Windows' NT's "User Manager" the changes that *RBAC Manager* has made to the Windows NT Security Database may be inspected. The screens below show that the users that were made role members have become members of the underlying groups. Three examples (Accounts Payable, Accounts Receivable, Administration) are shown below:



Local Group Properties

Group Name: Accounts Payable

Description:

Members:
Marsha
Nathan

Local Group Properties

Group Name: Accounts Receivable

Description:

Members:
Allan
Geoff

Local Group Properties

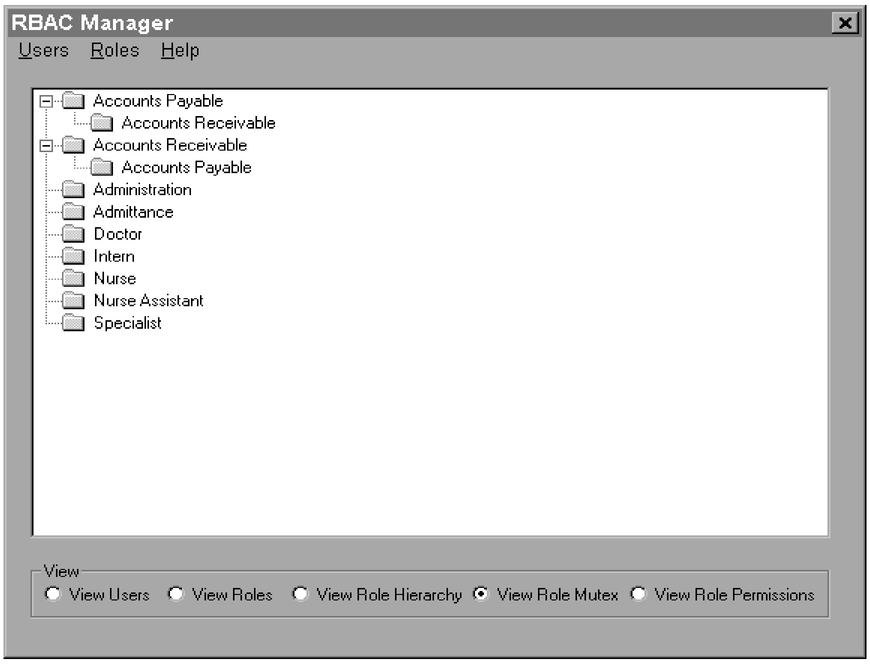
Group Name: Administration

Description:

Members:
Carolyn
Helen
Steve

Assigning Role Mutex

After assigning the roles described in the *Role Mutex* table as mutually exclusive, the *RBACManager Role Mutex View* resembles the following screen:



```
⊟ Accounts Payable
   └ Accounts Receivable
⊟ Accounts Receivable
   └ Accounts Payable
  Administration
  Admittance
  Doctor
  Intern
  Nurse
  Nurse Assistant
  Specialist
```

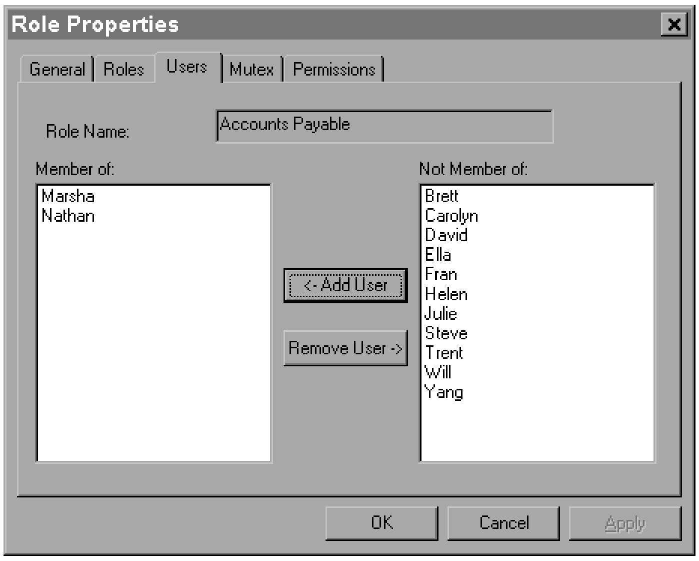
View: ○ View Users ○ View Roles ○ View Role Hierarchy [✓ View Role Mutex] ○ View Role Permissions

Defining roles as mutually exclusive does not alter the underlying Windows NT security mechanisms.

Once roles have been made mutually exclusive certain restrictions are placed on these roles. In particular, a user, role, or file permission cannot be assigned to both the mutually exclusive roles. This is illustrated below by showing that users of the *Accounts Payable* role cannot be assigned to the mutually exclusive *Accounts Receivable* role.

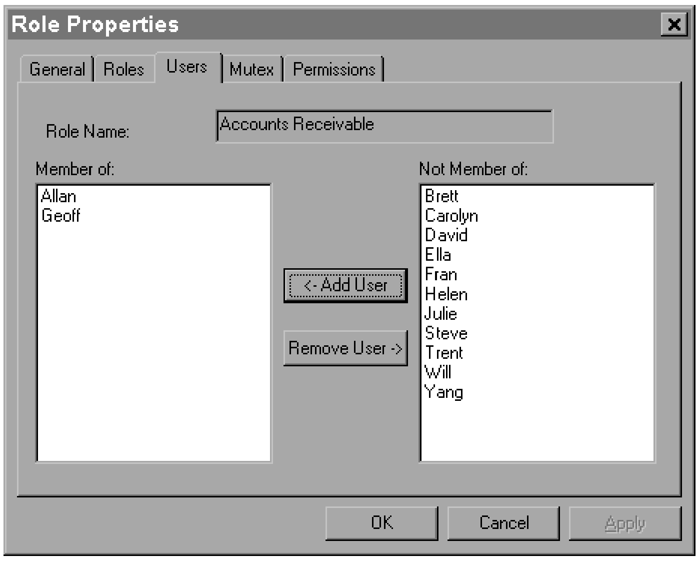
The bottom left screen shows that *Martha* and *Nathan* are members of the Accounts Payable role.

The bottom right screen shows that neither *Martha* nor *Nathan* can be assigned to the *Accounts Receivable* role as they have already been assigned to the *Accounts Payable* role which is mutually exclusive. That is, *Martha* or *Nathan* are not displayed so they cannot be selected.



Role Name: Accounts Payable

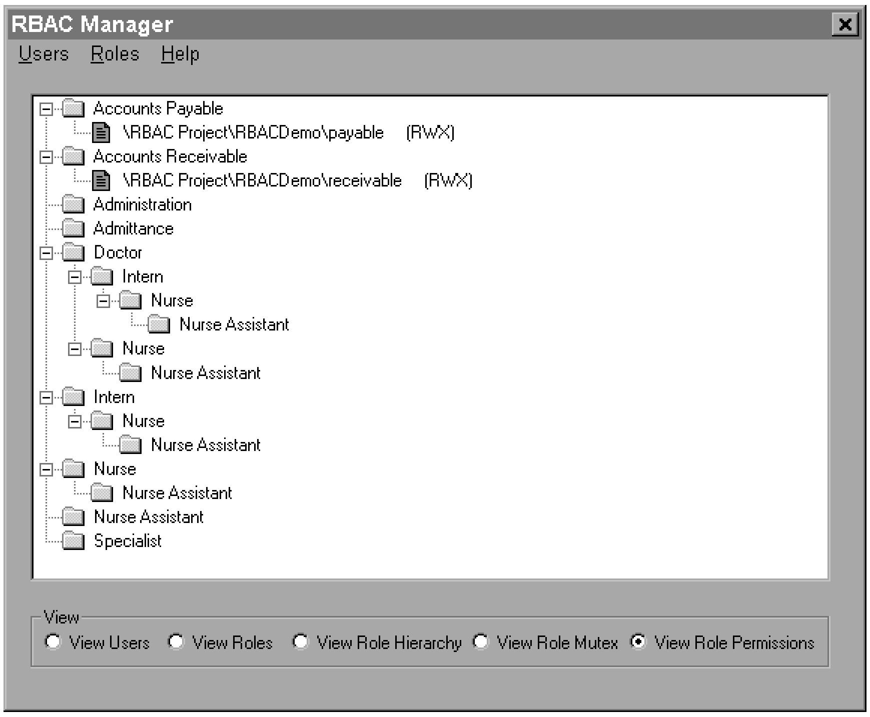
Member of:	Not Member of:
Marsha	Brett
Nathan	Carolyn
	David
	Ella
	Fran
	Helen
	Julie
	Steve
	Trent
	Will
	Yang



Role Name: Accounts Receivable

Member of:	Not Member of:
Allan	Brett
Geoff	Carolyn
	David
	Ella
	Fran
	Helen
	Julie
	Steve
	Trent
	Will
	Yang

This screen shows that the file '*\RBAC Project\RBACDemo\payable'* has been defined with file permissions linked to the *Accounts Payable* role.

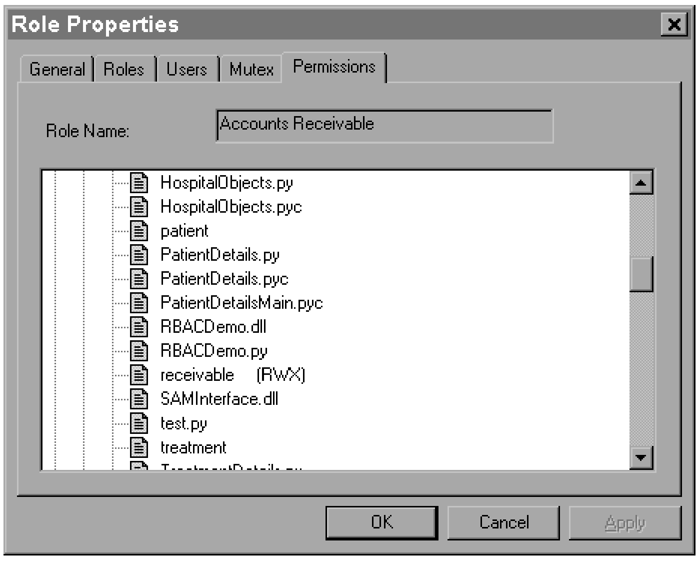

RBAC Manager

Users Roles Help

```
Accounts Payable
    \RBAC Project\RBACDemo\payable    (RWX)
Accounts Receivable
    \RBAC Project\RBACDemo\receivable    (RWX)
Administration
Admittance
Doctor
    Intern
        Nurse
            Nurse Assistant
    Nurse
        Nurse Assistant
Intern
    Nurse
        Nurse Assistant
Nurse
    Nurse Assistant
Nurse Assistant
Specialist
```

View: View Users, View Roles, View Role Hierarchy, View Role Mutex, [✓ View Role Permissions]

Mutually exclusive roles cannot share common file permissions. That is, '*\RBAC Project\RBACDemo\payable'* cannot be allocated the *Accounts Receivable* role as it has been allocated the mutually exclusive role *Accounts Payable*. Once again, it simply is not displayed, preventing it being selected.


Role Properties

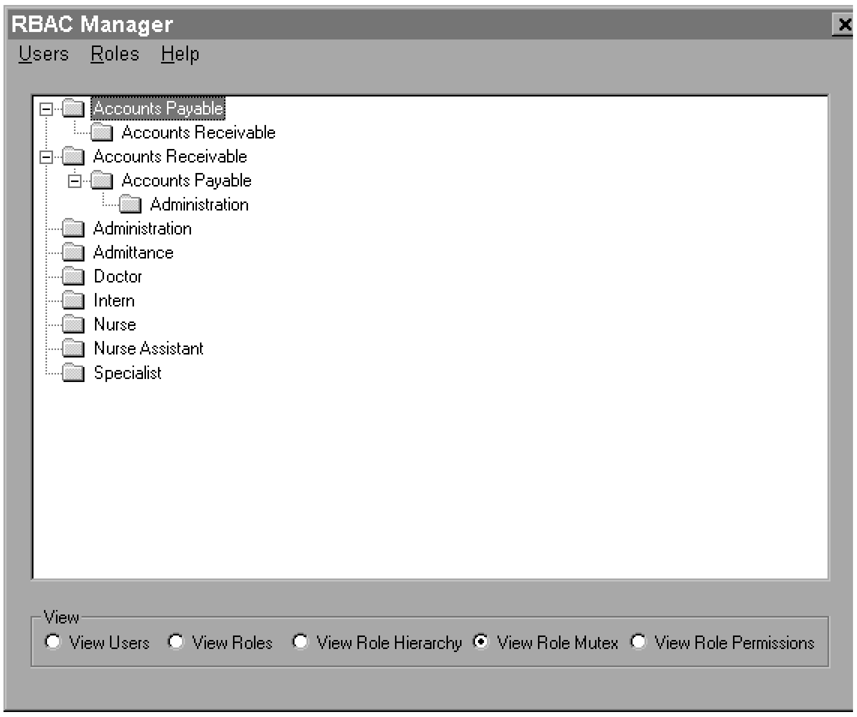
General | Roles | Users | Mutex | Permissions

Role Name: Accounts Receivable

```
HospitalObjects.py
HospitalObjects.pyc
patient
PatientDetails.py
PatientDetails.pyc
PatientDetailsMain.pyc
RBACDemo.dll
RBACDemo.py
receivable    (RWX)
SAMInterface.dll
test.py
treatment
```

OK | Cancel | Apply

It is worth noting that role members of mutually exclusive roles also become mutually exclusive. For illustration purposes, *Administration* has been made a member of *Account Payable* as shown here:


RBAC Manager

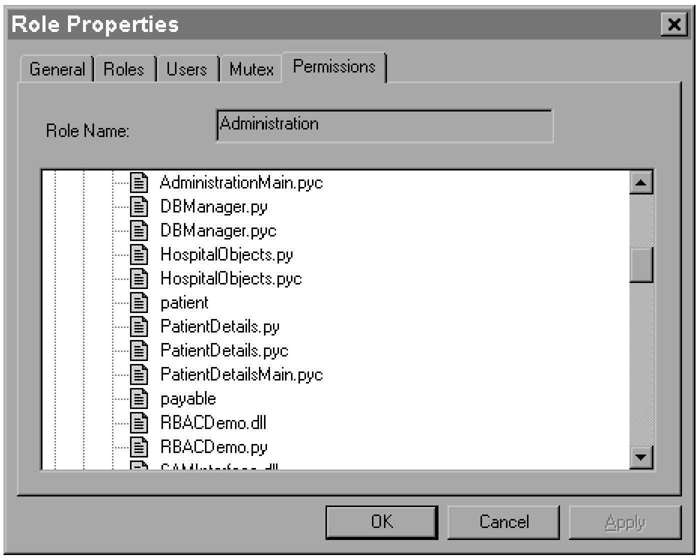
Users Roles Help

```
Accounts Payable
    Accounts Receivable
Accounts Receivable
    Accounts Payable
        Administration
Administration
Admittance
Doctor
Intern
Nurse
Nurse Assistant
Specialist
```

View: View Users, View Roles, View Role Hierarchy, [✓ View Role Mutex], View Role Permissions

In this case, *Administration also* becomes **mutually exclusive** with *Account Receivable* **by transitivity**.

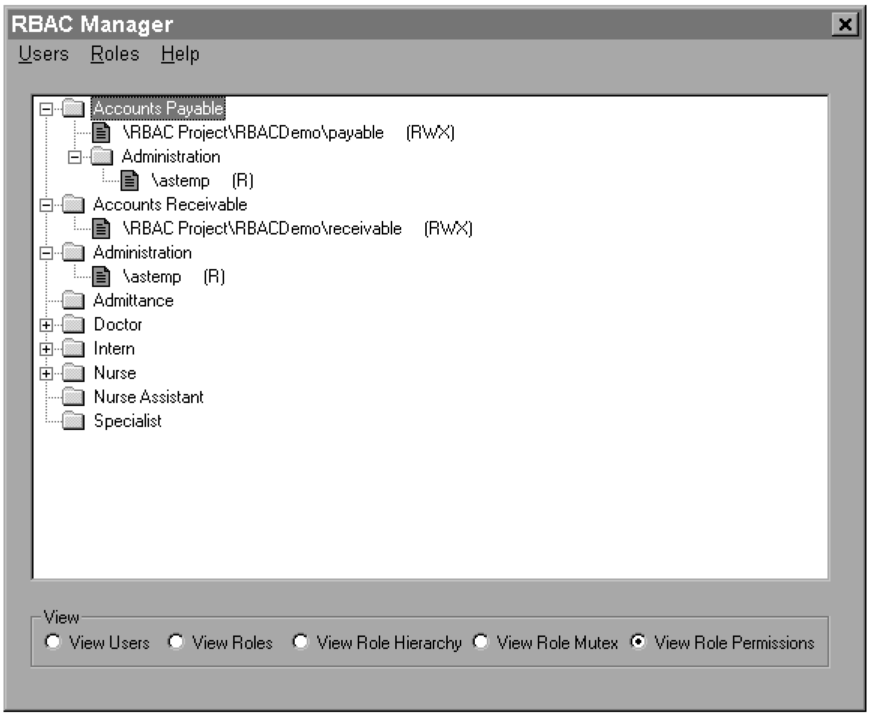
This is illustrated in this diagram by the fact that the file permissions previously assigned to the *Accounts Receivable* role (‘*\RBAC Project\RBACDemo\receivable’)* cannot be allocated the *Administration* role.



Role Name: Administration

AdministrationMain.pyc
DBManager.py
DBManager.pyc
HospitalObjects.py
HospitalObjects.pyc
patient
PatientDetails.py
PatientDetails.pyc
PatientDetailsMain.pyc
payable
RBACDemo.dll
RBACDemo.py

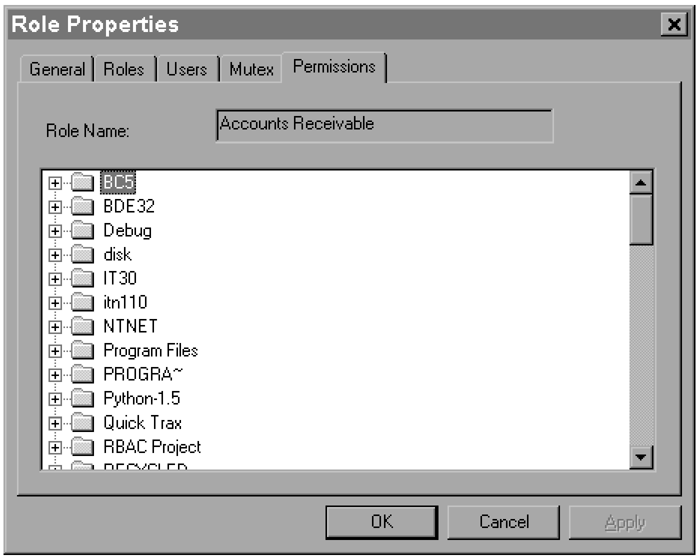
Likewise, if we assign the file ‘*\astemp*’ to the *Administration* role as shown:



```
Accounts Payable
    \RBAC Project\RBACDemo\payable    (RWX)
    Administration
        \astemp    (R)
Accounts Receivable
    \RBAC Project\RBACDemo\receivable    (RWX)
Administration
    \astemp    (R)
Admittance
Doctor
Intern
Nurse
Nurse Assistant
Specialist
```

View Users / View Roles / View Role Hierarchy / View Role Mutex / [✓ View Role Permissions]

the *Accounts Receivable* role cannot be assigned the file ‘*\astemp*’, which this screen illustrates:

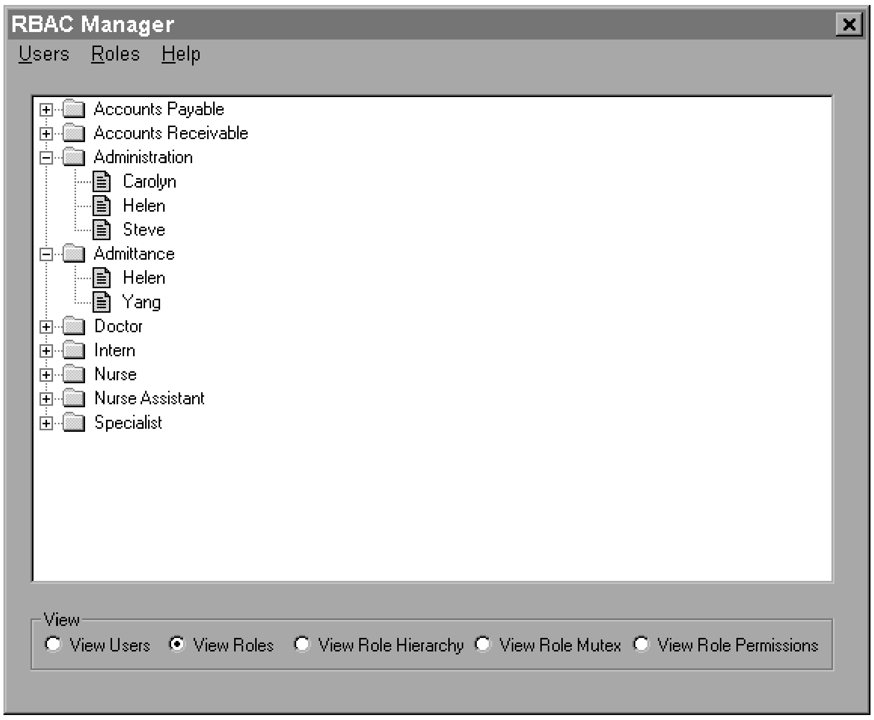


Role Name: Accounts Receivable

BC5
BDE32
Debug
disk
IT30
itn110
NTNET
Program Files
PROGRA~
Python-1.5
Quick Trax
RBAC Project

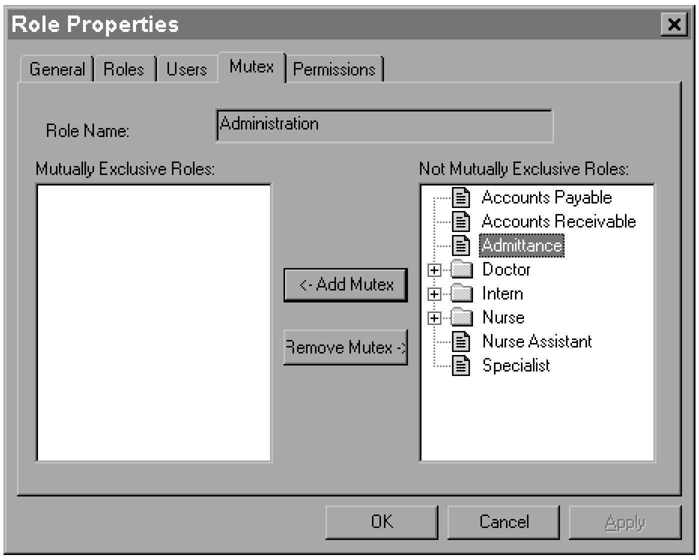
It is also worth noting that it is not possible to define two roles as mutually exclusive if it will violate the role's current users, roles and permissions.

For example, in the following illustrations, if we attempt to define the Administration role to be mutually exclusive with the Admittance role *RBACManager* will report an error and not allow the role to be defined as mutually exclusive.



- Accounts Payable
- Accounts Receivable
- Administration
  - Carolyn
  - Helen
  - Steve
- Admittance
  - Helen
  - Yang
- Doctor
- Intern
- Nurse
- Nurse Assistant
- Specialist

Here, we attempt to make the *Admittance* role a member of the *Administration* role.

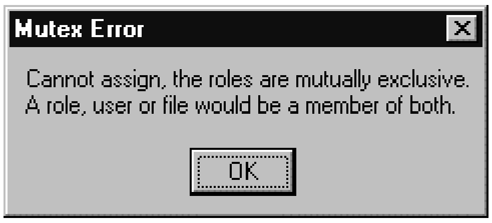


Role Name: Administration

Mutually Exclusive Roles:

Not Mutually Exclusive Roles:
- Accounts Payable
- Accounts Receivable
- Admittance
- Doctor
- Intern
- Nurse
- Nurse Assistant
- Specialist

The attempted operation will result in *RBACManager* reporting the following error, (since user Helen is in both roles) and not allow the operation to be executed.

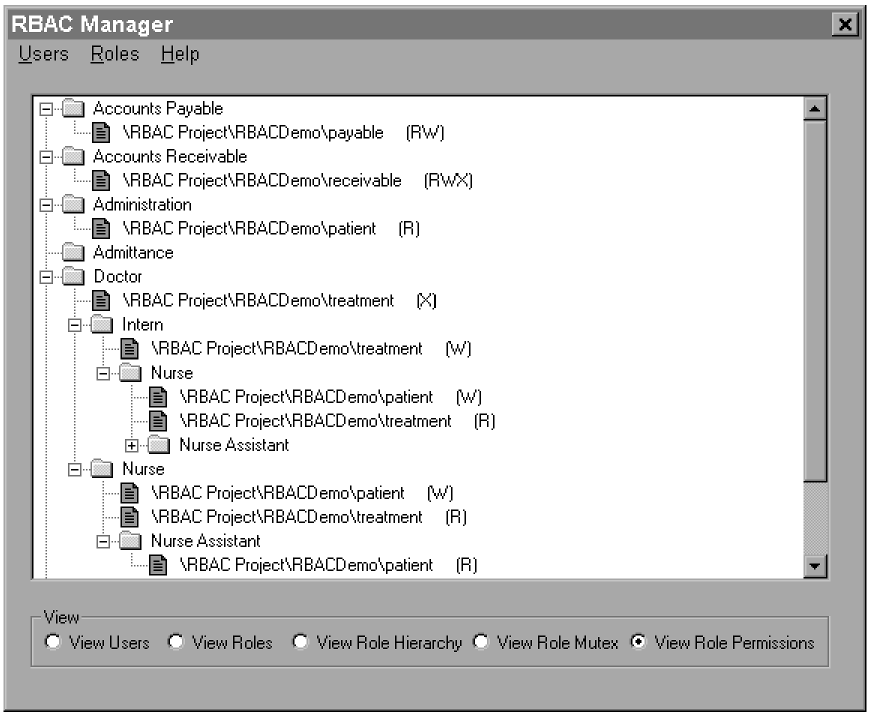


Cannot assign, the roles are mutually exclusive.
A role, user or file would be a member of both.

Assigning Permissions

The *RBACManager Permissions View* resembles this screen after assigning the permissions described in the *Role Permissions* table.

Note particularly the permissions assigned to the various instances of the patient and treatment files. These will accumulate since roles accumulate permissions from their children in the hierarchy.

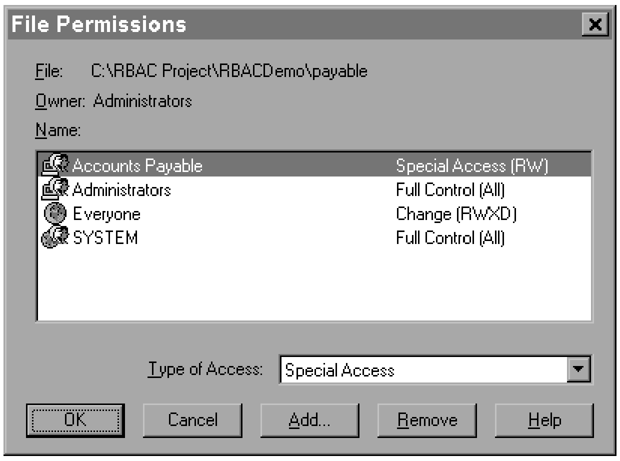


```
Accounts Payable
    \RBAC Project\RBACDemo\payable    (RW)
Accounts Receivable
    \RBAC Project\RBACDemo\receivable    (RWX)
Administration
    \RBAC Project\RBACDemo\patient    (R)
Admittance
Doctor
    \RBAC Project\RBACDemo\treatment    (X)
    Intern
        \RBAC Project\RBACDemo\treatment    (W)
        Nurse
            \RBAC Project\RBACDemo\patient    (W)
            \RBAC Project\RBACDemo\treatment    (R)
            Nurse Assistant
    Nurse
        \RBAC Project\RBACDemo\patient    (W)
        \RBAC Project\RBACDemo\treatment    (R)
        Nurse Assistant
            \RBAC Project\RBACDemo\patient    (R)
```

View: View Users, View Roles, View Role Hierarchy, View Role Mutex, [✓ View Role Permissions]

RBACManager controls the file's access control by adding the role's underlying group to the file's access control list (ACL). The following diagram illustrates this by showing the files ACL.

This screen shows the ACL for the *payable* file,

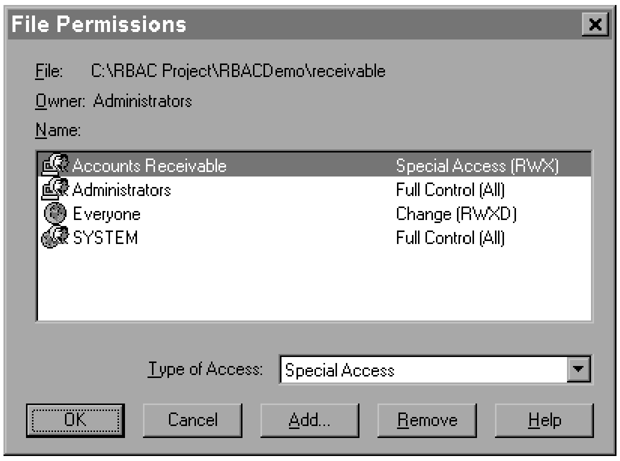


File: C:\RBAC Project\RBACDemo\payable
Owner: Administrators
Name:

Accounts Payable	Special Access (RW)
Administrators	Full Control (All)
Everyone	Change (RWXD)
SYSTEM	Full Control (All)

Type of Access: Special Access

and this screen shows the ACL for the *receivable* file.



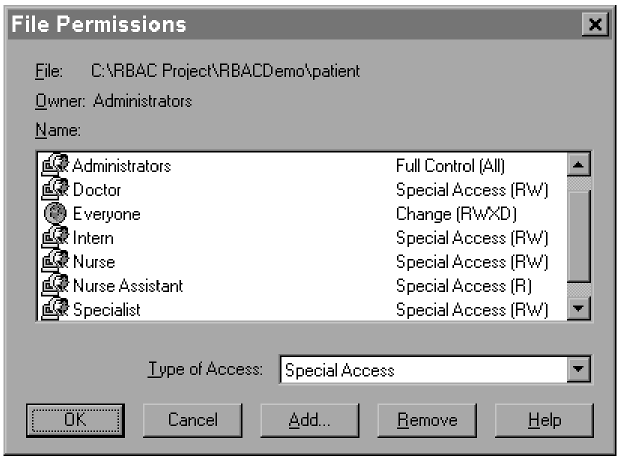
File: C:\RBAC Project\RBACDemo\receivable
Owner: Administrators
Name:

Accounts Receivable	Special Access (RWX)
Administrators	Full Control (All)
Everyone	Change (RWXD)
SYSTEM	Full Control (All)

Type of Access: Special Access

These two screens demonstrate **that roles accumulate permissions from children in the hierarchy**.

The first screen shows the ACL for the *patient* file.



File Permissions

File: C:\RBAC Project\RBACDemo\patient

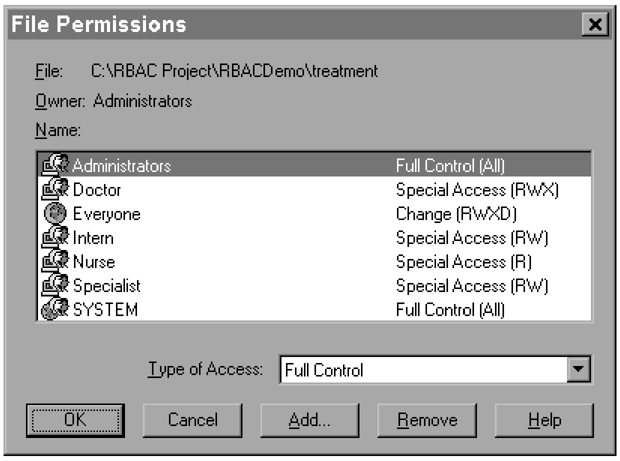
Owner: Administrators

Name:

Administrators	Full Control (All)
Doctor	Special Access (RW)
Everyone	Change (RWXD)
Intern	Special Access (RW)
Nurse	Special Access (RW)
Nurse Assistant	Special Access (R)
Specialist	Special Access (RW)

Type of Access: Special Access

The second screen shows the ACL for the *treatment* file.



File Permissions

File: C:\RBAC Project\RBACDemo\treatment

Owner: Administrators

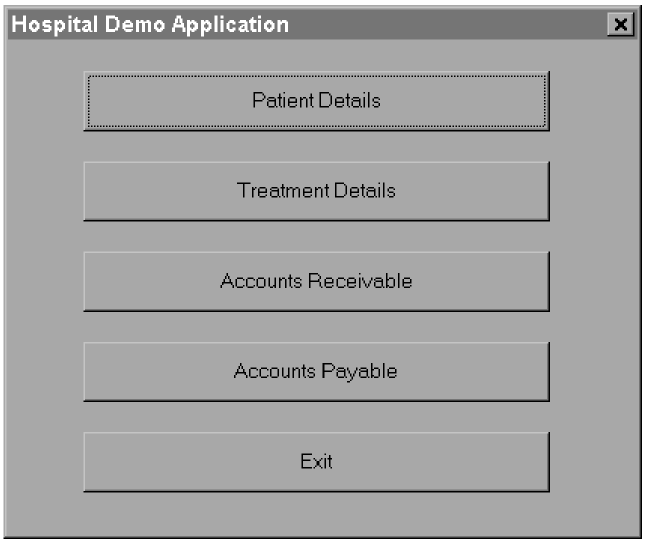
Name:

Administrators	Full Control (All)
Doctor	Special Access (RWX)
Everyone	Change (RWXD)
Intern	Special Access (RW)
Nurse	Special Access (R)
Specialist	Special Access (RW)
SYSTEM	Full Control (All)

Type of Access: Full Control

RBAC$_{M}$ Demonstration – Phase 2: Execution of Prototype Application

The main menu of the prototype application developed to demonstrate the successful security administration by *RBACManager* is shown here.



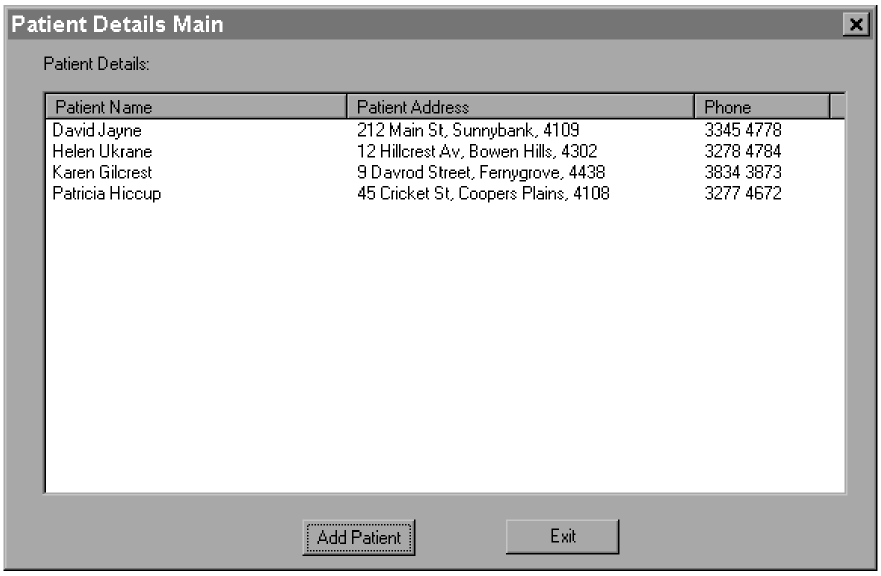
As can be seen the menu contains 4 options that access the files specified below:

Option	File
Patient Details	\RBAC Project\RBACDemo\patient
Treatment Details	\RBAC Project\RBACDemo\treatment
Accounts Receivable	\RBAC Project\RBACDemo\receivable
Accounts Payable	\RBAC Project\RBACDemo\payable

Patient Details Menu Option

The Patient Details menu option displays the following screen:

A user requires *READ* access to the *patient* file to access this option.

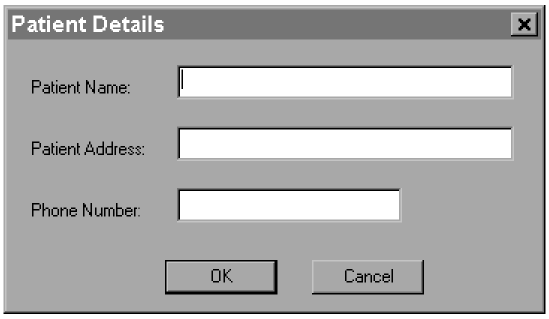


Patient Details:

Patient Name	Patient Address	Phone
David Jayne	212 Main St, Sunnybank, 4109	3345 4778
Helen Ukrane	12 Hillcrest Av, Bowen Hills, 4302	3278 4784
Karen Gilcrest	9 Davrod Street, Fernygrove, 4438	3834 3873
Patricia Hiccup	45 Cricket St, Coopers Plains, 4108	3277 4672

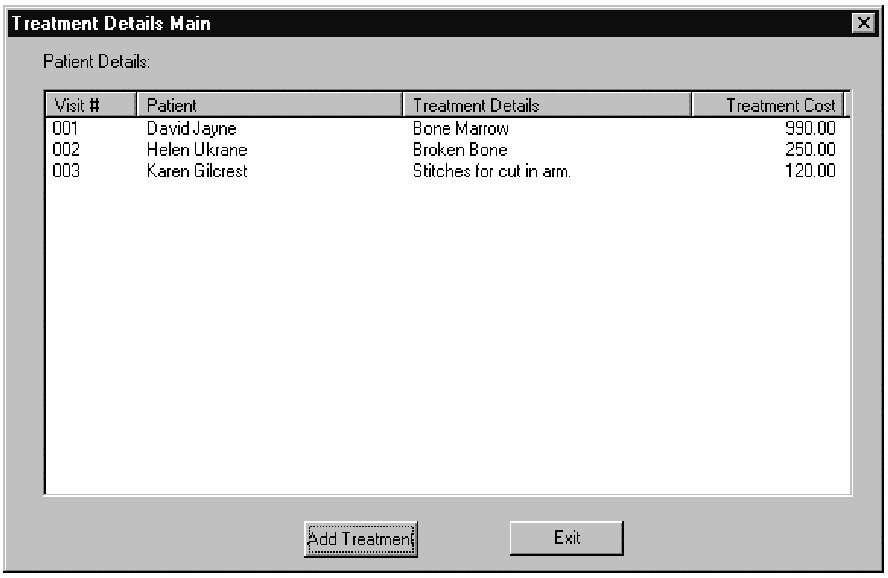
The Add Patient button displays the following screen:

A user requires *WRITE* access to the *patient* file to add a patient.



Treatment Details Menu Option

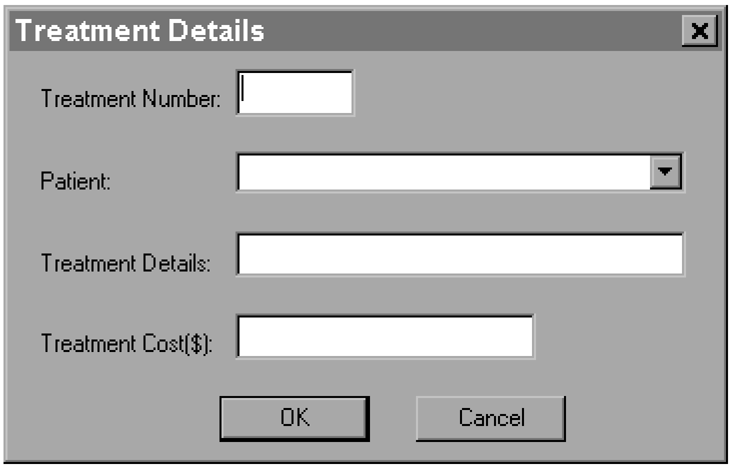
The Treatment Details menu option displays the following screen:



Visit #	Patient	Treatment Details	Treatment Cost
001	David Jayne	Bone Marrow	990.00
002	Helen Ukrane	Broken Bone	250.00
003	Karen Gilcrest	Stitches for cut in arm.	120.00

A user requires *READ* access to the *treatment* file to access this option.

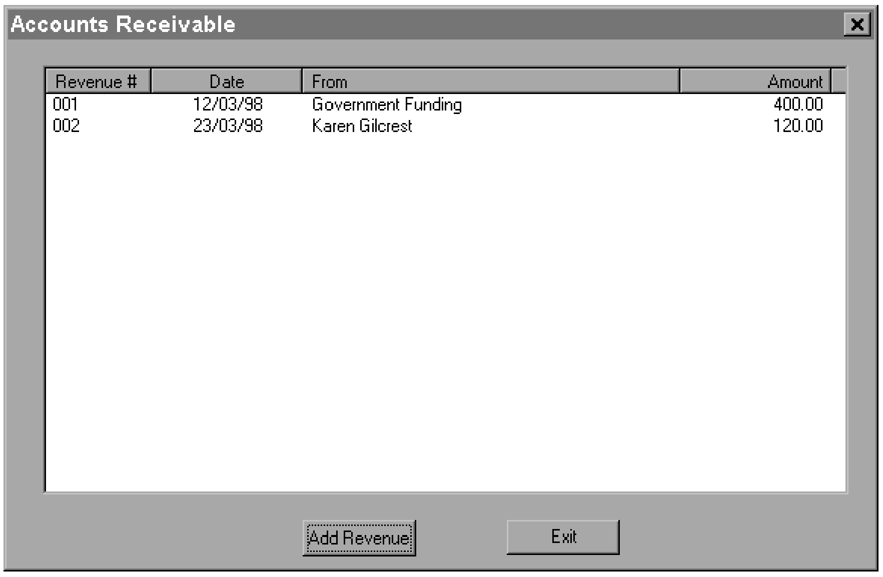
The Add Treatment button displays the following screen:



A user requires *WRITE* access to the *treatment* file **and** *READ* access to the *patient* file to add treatment details.

Accounts Receivable Menu Option

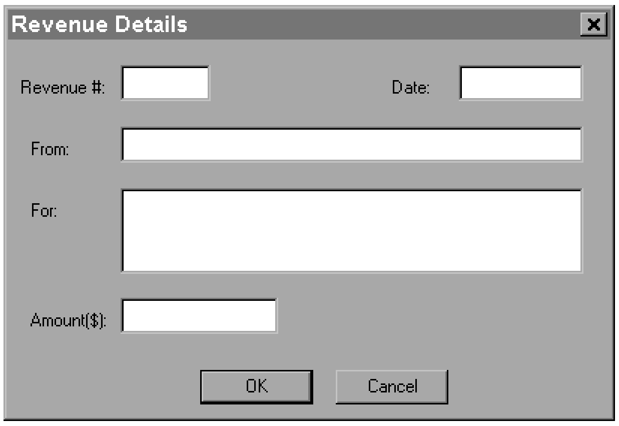
The Accounts Receivable menu option displays the following screen:



Revenue #	Date	From	Amount
001	12/03/98	Government Funding	400.00
002	23/03/98	Karen Gilcrest	120.00

A user requires *READ* access to the *receivable* file to access this option.

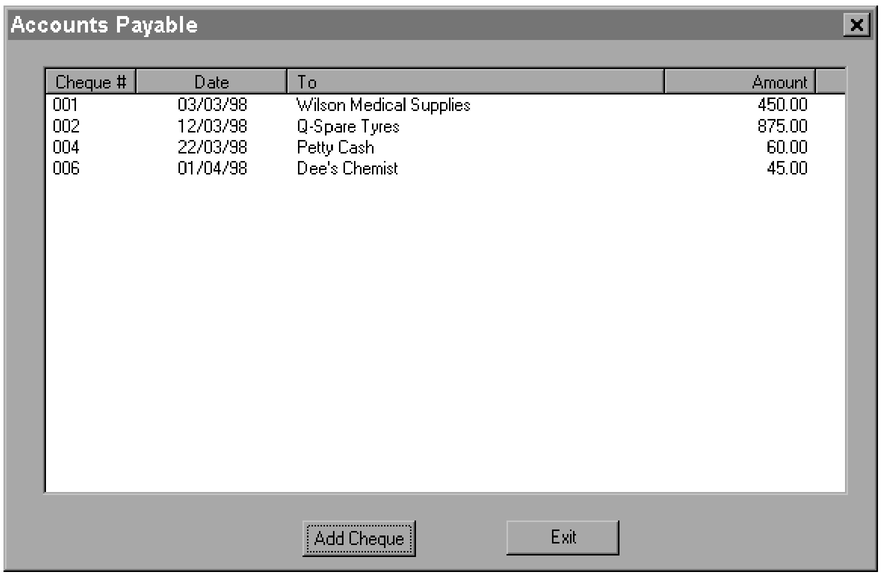
The Add Revenue button displays the following screen:



A user requires *WRITE* access to the *receivable* file to add revenue details.

Accounts Payable Menu Option

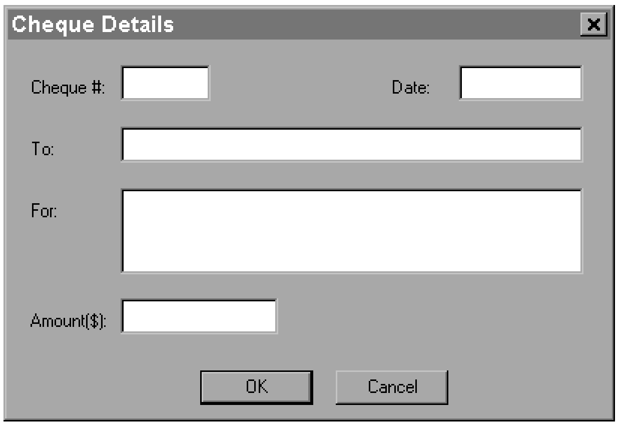
The Accounts Payable menu option displays the following screen:



Cheque #	Date	To	Amount
001	03/03/98	Wilson Medical Supplies	450.00
002	12/03/98	Q-Spare Tyres	875.00
004	22/03/98	Petty Cash	60.00
006	01/04/98	Dee's Chemist	45.00

A user requires *READ* access to the *payable* file to access this option.

The Add Cheque button displays the following screen:



A user requires *WRITE* access to the *payable* file to add cheque details.

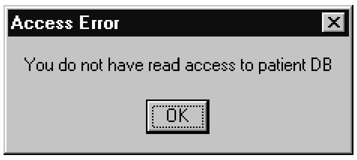
Test Cases

The following test cases illustrate that *RBACManager* has enforced the desired security.

Marsha (Accounts Payable)

Attempt to access Patient Details.

Denied as specified.

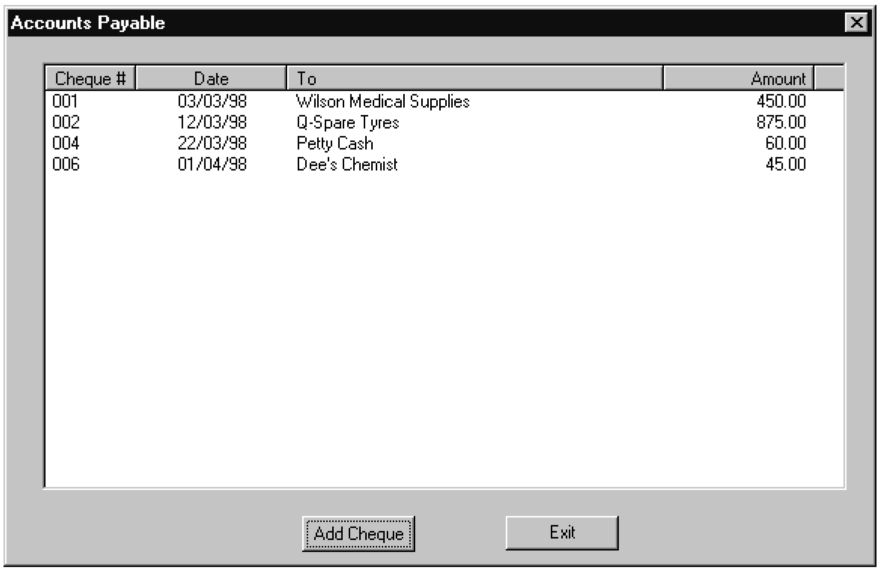


```
Access Error

You do not have read access to patient DB
```

Attempt to access Account Payable.

Allowed as specified.



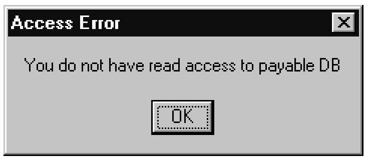
Accounts Payable

Cheque #	Date	To	Amount
001	03/03/98	Wilson Medical Supplies	450.00
002	12/03/98	Q-Spare Tyres	875.00
004	22/03/98	Petty Cash	60.00
006	01/04/98	Dee's Chemist	45.00

Allan (Accounts Receivable)

Attempt to Access Accounts Payable.

Denied as specified.

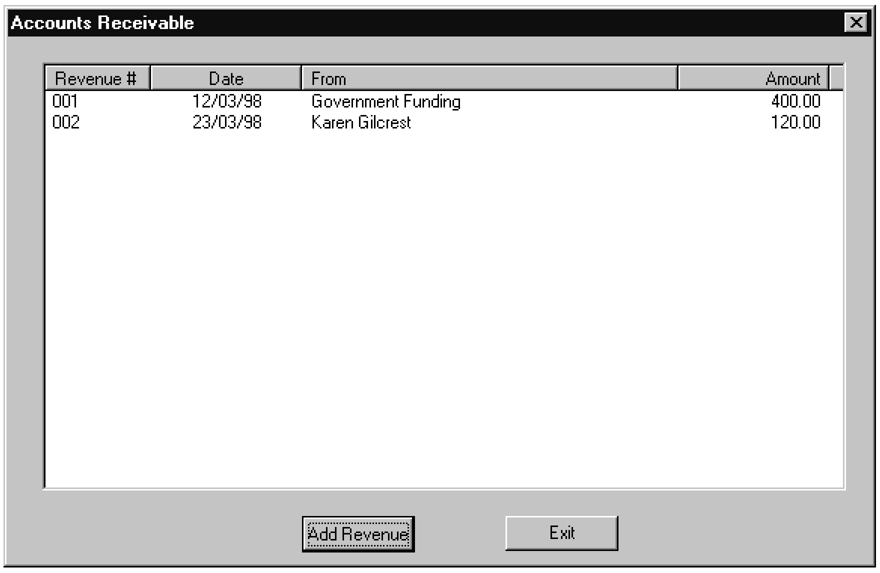


```
Access Error

You do not have read access to payable DB
```

Attempt to access accounts receivable.

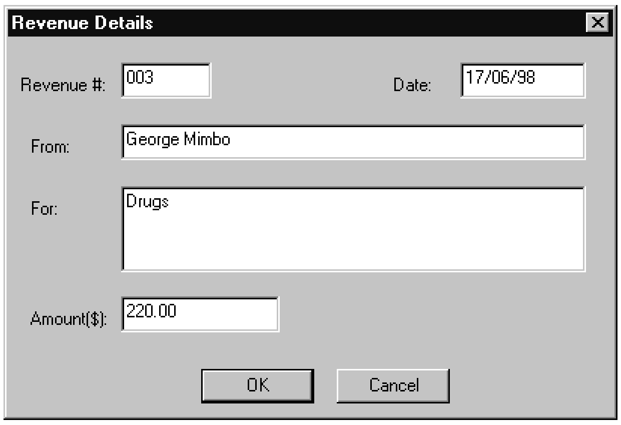
Allowed as specified.



Accounts Receivable

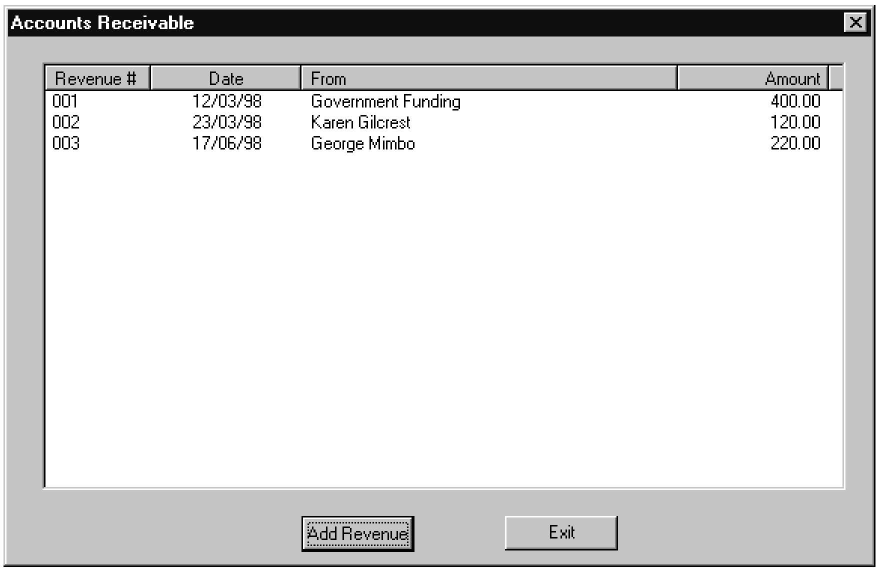
Revenue #	Date	From	Amount
001	12/03/98	Government Funding	400.00
002	23/03/98	Karen Gilcrest	120.00

Attempt to add revenue details.


Revenue Details

Revenue #:	003	Date:	17/06/98
From:	George Mimbo		
For:	Drugs		
Amount($):	220.00		

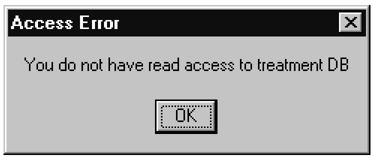
Successful as specified in the role permissions table.


Accounts Receivable

Revenue #	Date	From	Amount
001	12/03/98	Government Funding	400.00
002	23/03/98	Karen Gilcrest	120.00
003	17/06/98	George Mimbo	220.00

Fran (Nurse Assistant)

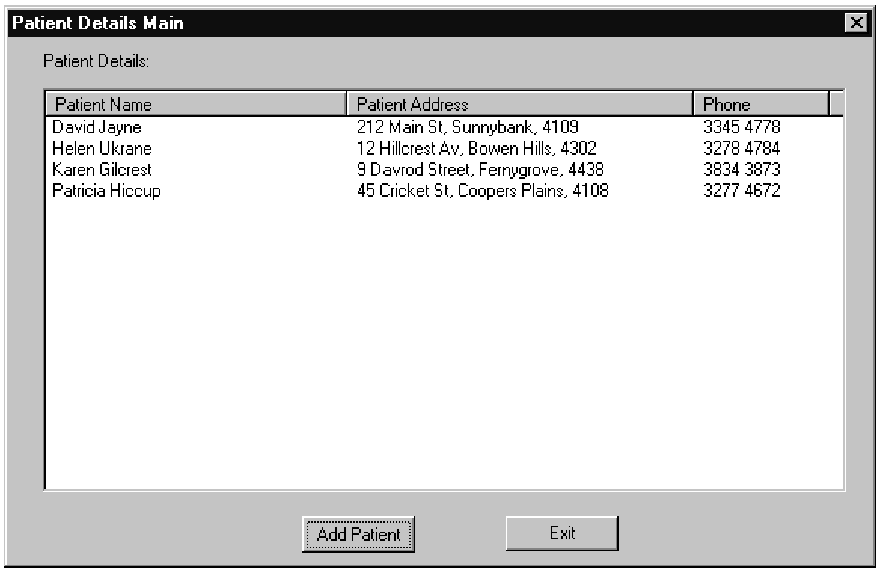
Attempt to access treatment database. Denied, as Fran only has access to the patient database.


Access Error

You do not have read access to treatment DB

Attempt to access patient details.

Allowed as specified.

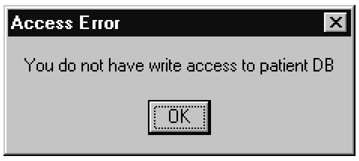

Patient Details Main

Patient Details:

Patient Name	Patient Address	Phone
David Jayne	212 Main St, Sunnybank, 4109	3345 4778
Helen Ukrane	12 Hillcrest Av, Bowen Hills, 4302	3278 4784
Karen Gilcrest	9 Davrod Street, Fernygrove, 4438	3834 3873
Patricia Hiccup	45 Cricket St, Coopers Plains, 4108	3277 4672

Attempt to add patient details.

Denied since Fran only has read access, not write.

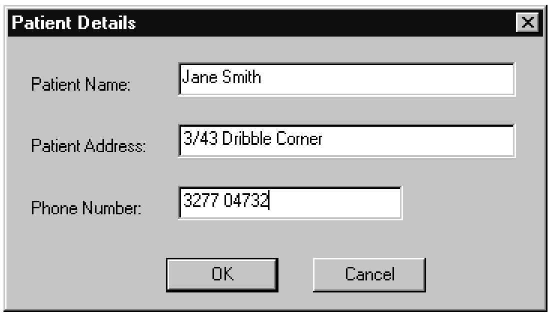


Access Error

You do not have write access to patient DB

David (Nurse)

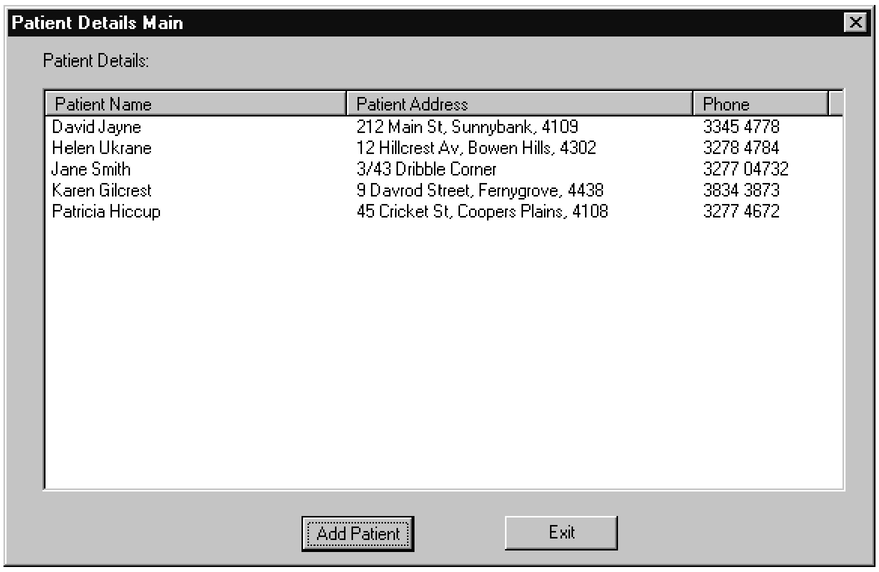
Attempt to Add Patient Details



Patient Details

Patient Name:	Jane Smith
Patient Address:	3/43 Dribble Corner
Phone Number:	3277 04732

Allowed since David has access to the patients database.



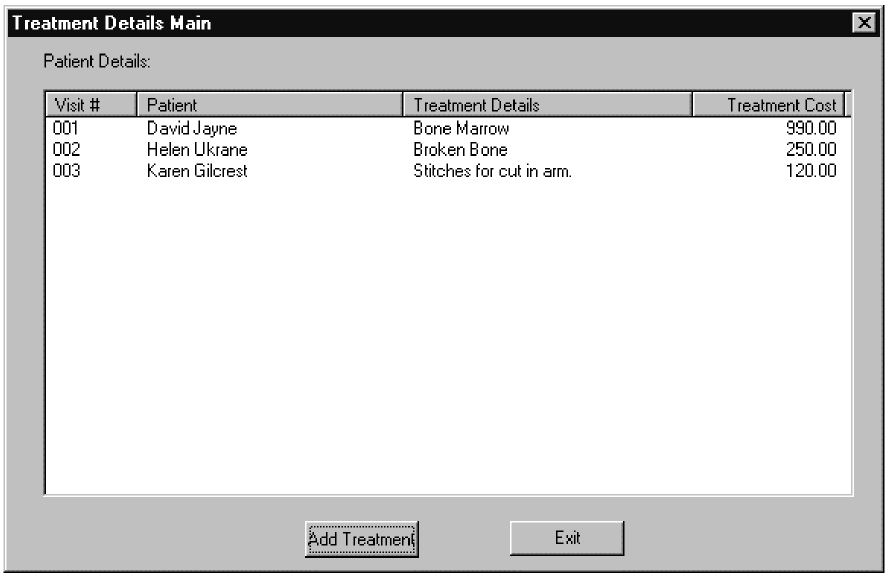
Patient Details Main

Patient Details:

Patient Name	Patient Address	Phone
David Jayne	212 Main St, Sunnybank, 4109	3345 4778
Helen Ukrane	12 Hillcrest Av, Bowen Hills, 4302	3278 4784
Jane Smith	3/43 Dribble Corner	3277 04732
Karen Gilcrest	9 Davrod Street, Fernygrove, 4438	3834 3873
Patricia Hiccup	45 Cricket St, Coopers Plains, 4108	3277 4672

Attempt to access treatment details:

Also allowed since David has access to treatment database.



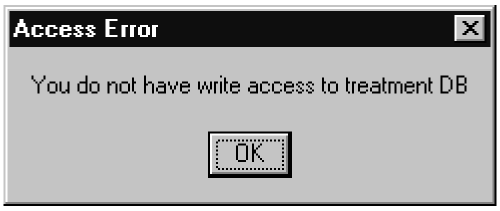
Treatment Details Main

Patient Details:

Visit #	Patient	Treatment Details	Treatment Cost
001	David Jayne	Bone Marrow	990.00
002	Helen Ukrane	Broken Bone	250.00
003	Karen Gilcrest	Stitches for cut in arm.	120.00

Attempt to add treatment details:

Not allowed since David only has read access, not write.

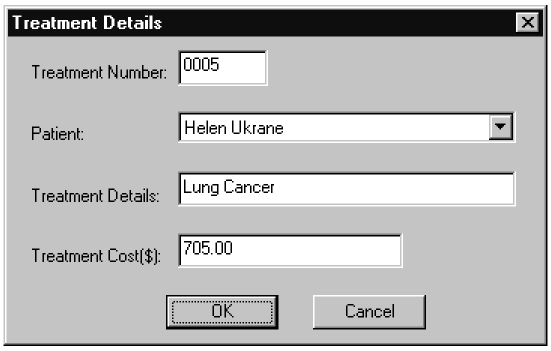


Access Error

You do not have write access to treatment DB

Ella (Doctor)

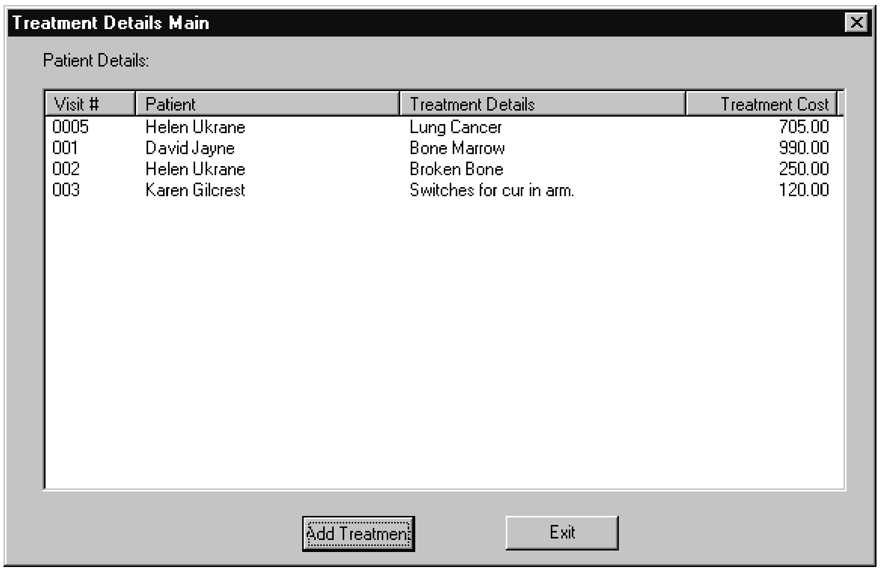
Attempt to add treatment details:



Treatment Details

Treatment Number:	0005
Patient:	Helen Ukrane
Treatment Details:	Lung Cancer
Treatment Cost($):	705.00

Allowed as specified.



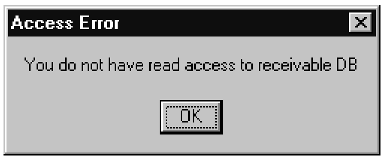
Treatment Details Main

Patient Details:

Visit #	Patient	Treatment Details	Treatment Cost
0005	Helen Ukrane	Lung Cancer	705.00
001	David Jayne	Bone Marrow	990.00
002	Helen Ukrane	Broken Bone	250.00
003	Karen Gilcrest	Switches for cur in arm.	120.00

Attempt to access account receivable

Denied as specified.



Access Error

You do not have read access to receivable DB

<u>RBAC$_M$ Summary</u>

The RBAC$_M$ administration tool, RBACManager, was successfully implemented in Windows NT. This tool proved that an RBAC framework could be implemented under an operating system such as Windows NT that supports access control lists (ACL). The implementation provided an insight into Windows NT and RBAC$_M$ and has provided a solid foundation for the future research that is discussed later in this paper. Some issues arose from this experimental implementation. These are discussed below.

RBAC$_M$ Administration Tool Design

The RBAC$_M$ administration tool uses it's own database to store the RBAC configuration. This database is separate from the Windows NT security mechanism. The RBAC$_M$ administration tool simply manages the configuration (role hierarchies, constraints, etc) and translates the RBAC configuration into the underlying Windows NT security mechanisms. For example, a role is translated into a group.

Further research could investigate aligning the information stored in the $RBAC_M$ administration tool and the underlying Windows NT security mechanism. This may provide a more cohesive, extendable solution, if technically possible.

Concurrent Access

One issue not addressed in the implementation is the inevitable simultaneous access by multiple processes. In particular, if another process tries to access a file's ACL while the $RBAC_M$ administration tool is updating it, there could (more than likely) be disastrous effect. Worst case scenario could be complete loss of the file.

This was outside the scope of this initial version. However, there is a definite need to serialize the access to the ACL to ensure the tool is sufficiently robust to execute in a distributed environment.

Everyone Group

Another problem encountered was that every file created contained the “everyone” group in its ACL. (In Windows NT the “everyone” group is a special (super)group that includes all other default Windows NT groups and any local groups and therefore the members of each of those groups). This allows anyone to access the file although the $RBAC_M$ administration tool had not explicitly granted access.

All $RBAC_M$ created files contained the “everyone” group in the ACL since the file was created under the root directory (C:\) which is a container object. This means that every file created in the container object inherits the container's ACL. This will require further investigation to provide a secure system that is fully controlled by the $RBAC_M$ administration tool.

Technical Highlights

The goal behind the $RBAC_M$ implementation was a detailed investigation into Windows NT security and the RBAC paradigm to discover the best approach for integrating an RBAC framework. This led to some challenging and interesting technical achievements during this implementation. Some of these are presented below.

Application Level vs. System Level

In operating systems other than Windows NT it is quite common to find user databases and passwords lists for individual applications. These multiple databases are maintained to restrict access by a subset of users to the different functions of an application.

This is illustrated in Windows and DOS operating systems where there are normally many lists of passwords defined for many different purposes. For example a user may be required to provide a password when logging on to each domain (or File Server in NetWare) on the network, another to access e-mail, and yet another to get back into the system after the screen saver has kicked in.

In Windows NT, additional passwords would be redundant as well as unnecessary, and would probably prevent these applications from selling into a C2-secure environment. Instead, system administrators simply create groups with the required restrictions to preclude unprivileged users. The application is then able to use the **Win32 security API** to determine whether the current user qualifies to perform certain operations throughout the application.

This was the approach adopted for the implementation of $RBAC_M$.

Impersonations

In Windows NT, the security levels are assigned to users and not the processes or threads that execute. Therefore, the security abilities for a process or thread change as different users (with different security levels) execute them. This may be permissable for standalone applications as each user executes the application in their own address space.

However, development of a client-server application requires great care when dealing with access to secure objects. As the server portion of the application is under control of the system which is likely to have extended privileges, a request from a client may result in the server returning data to which the client does not have access. This is a breach of security.

To overcome such problems Window NT provides a concept known as impersonation. Impersonation in a client-server application in general, and in Windows NT networks in particular, is very widely used. Impersonation is the act of taking the identity of another user account and acting in its security context, akin to the UNIX *suid* feature. Therefore, in client-server applications, impersonations allow servers to access data on behalf of privileged clients by assuming the security level of the client.

Furthermore, any process in the Windows NT system may try to impersonate any other process. Such actions are under the control of the operating system for security reasons, otherwise there would be no security at all.

Also, some Win32 functions require impersonation tokens (instead of the access token) as a parameter. For example, $RBAC_M$ required calling the AccessCheck() function to determine if a user has particular access to an object. The AccessCheck() function requires an impersonation token of the currently logged on user. To get an impersonation token in this situation, you have to impersonate yourself. Here's how you do that:

1. Call ImpersonateSelf() to begin the impersonation.
2. Call OpenThreadToken() to get a HANDLE to the impersonation token. You must use OpenThreadToken() because OpenProcessToken() returns the original token of the process.
3. Do whatever you need with the token. In this particular case, call AccessCheck().
4. Call RevertToSelf() to end the impersonation.

Groups

A group is a useful mechanism which helps to simplify the administration of users on a network. A group is a "named collection of users". A group is assigned a SID just as an individual user. By using a group's SID in a discretionary access control list of a security descriptor, you may deny or allow access for all users in the group. Windows NT has two types of groups: Global and Local groups. A global group is a named collection of user accounts that is visible to any computer participating in a domain. A local group only exists on an individual computer.

Windows NT local groups can contain global groups as members. However, Windows NT global groups cannot have local or global groups as members. Unfortunately, this adds extra complexities when dealing with the role hierarchies of an RBAC framework. If the operating system allows groups to be members of groups the role hierarchy could be handled by the underlying operating system. However, $RBAC_M$ required the $RBAC_M$ program to accumulate the permissions from roles lower in the hierarchy to determine the access level to assign to a file's ACL for the role's corresponding group. Also, if a lower lever role's permissions are changed all the roles higher in the hierarchy required the permissions to be re-calculated and each corresponding file's ACL must be updated. This introduces efficiency issues for large hierarchies.

Future Research

RBAC Issues

Although there is much agreement on the basic concepts and value of RBAC, a number of remaining issues still confront the RBAC community. Considerable research and work remains to develop solid theoretical and practical foundations in the area.

First and foremost the continuing evolution of RBAC needs to be closely monitored to ensure that industry proceeds in a common and consistent direction. Although at the time of the $RBAC_M$ implementation it was unknown if a common formal framework will be acceptable across the entire industry, there is a clear need to define and guide the evolution of a reference model (Ferraiolo 1996). This will also require careful consideration to ensure that the evolving RBAC aligns with other emerging concepts and models in computer industry such as the Internet, interoperable objects and software components, and workflow automation (Ferraiolo 1996). Subsequent to the development of $RBAC_{M,}$ an RBAC "Common Criteria" specification became available in September 1998.

Recent interest in RBAC has focused on integrating RBAC at the application level (Sandhu et al. 1996). Applications have been built with RBAC encoded within the application itself. Operating systems, however, provide little support for application-level use of RBAC. Therefore, a challenge facing the user community is identifying application-independent facilities that are sufficiently flexible, yet simple to implement and use, to support a wide range of application with minimal customization.

There also appears to be a lack of research relating to the management aspects of RBAC that needs to be addressed before the industry advances. In particular, the development of a systematic methodology that guide the analysis and design of an organization's RBAC configuration (role hierarchies, constraints, RBAC management in a unified framework) is one area requiring particular research attention (Sandhu 1996). There is also little discussion in the literature regarding the constraints applied within an RBAC environment. That is, the categorization and taxonomy of constraints, along with some measure of difficulty of enforcement.

Workflow Environments

It has been discovered that the currently accepted notion of RBAC is not ideally suited for the security needs of all organizations (Sandhu et al. 1996). More sophisticated models are required to control access in situations where sequences of operations need to be governed, such as workflow environments.

The completed research effort, $RBAC_M$, has provided a solid foundation for investigation into higher level models that are active in nature. Most well known access control models are considered to be passive in nature. These models do not distinguish between permission assignment and activation. Furthermore, passive security models are not capable of representing or considering any levels of context when processing an access operation on an object. It is expected that active security concepts to be an important area of future research and we believe they will influence the evolution of RBAC.

Although, RBAC has been identified as a security model that would be well suited in collaborative environments, such as workflow management systems, the passive and rigid nature of current RBAC models present problems that prevent a natural integration. In particular, current RBAC models do not allow fine-grain control of individual users in certain roles and on individual object instances. RBAC also provides no support for the context associated with collaborative tasks.

Further research will be conducted in this area and will initially involve a detailed examination of the suggested model proposed by Thomas (1997). Investigation of the recent work by Bertino, Ferrari & Atluri (1997) will also be conducted in the area such that a high level access control language/parser/interpreter could be defined and developed which is suitable for incorporation into modern operating systems such as Windows NT.

In summary, the overall aim and intent of the proposed research will be to investigate current RBAC models and the possible methods that can be applied to transform passive models into active models such that they can fulfil the current security requirements of collaborative environments.

Bibliography

Bertino, E., Ferrari, E. & Atluri, V. 1997, 'A Flexible Model Supporting the Specification and Enforcement of Role-based Authorizations in Workflow Management Systems', *Proceeding of the Second ACM Workshop on Role-Based Access Control*, ACM, November 1997, pp. 1-12.

Chen, F. & Sandhu, R. 1995, 'Constraints for Role Based Access Control', *Proceedings of the First ACM Workshop on Role-Based Access Control*, December 1995.

Davis, R. 1996, 'Win32 Network Programming', Addison-Wesley Developers Press, 1996.

Ferraiolo, D.F. 1996, 'An Introduction to Role-Based Access Control', *Internal Report, Computer Systems Laboratories NIST*, January 1996.

Ferraiolo, D. F. 1996, 'Toward a Common Framework for Role-Based Access Control', [Online] URL http://hissa.ncsl.nist.gov/rbac/ [Accessed 9 April 1998]

Hamilton, D. & Williams, M. 1996, 'Programming Windows NT 4 Unleashed', SAMS Publishing, 1996.
Kuhn, D. R. 1997, 'Mutual Exclusion as a Means of Implementing Separation of Duty in Role-Bases Access Control System', *Proceeding of the Second ACM Workshop on Role-Based Access Control*, ACM, November 1997, pp23-40.

Microsoft Press 1995, 'Microsoft Windows NT Resource Kit', Microsoft Press, USA

Okuntseff, N. 1997, 'Windows NT Security – Programming Easy-to-Use Security Options', Publishers Group West, CA

Sandhu, R. 1996, 'Report on the First ACM Workshop on Role-based Access Control', *Internal Report, Computer Systems Laboratories NIST*, March 1996.

Sandhu, R., Coyne, E. J., Feinstein, H. L. & Youman, C.E. 1996, 'Role-Based Access Control Models', *IEEE Computer*, February 1996, pp. 38-47.

Thomas, R. K. 1997, 'Team-based Access Control (TMAC): A Primitive for Applying Role-based Access Controls in Collaborative Environments', *Proceeding of the Second ACM Workshop on Role-Based Access Control*, ACM, November 1997, pp. 13-19.