OPC DCOM White Paper

Richard C. Harrison, Intellution Inc.

Abstract

For OPC implementations in general, it is expected that OPC Server vendors will take one of two approaches to networking:

- The client may connect to a local server which makes use of an existing proprietary network scheme. This approach will commonly be used by vendors who are adding OPC capability to an existing distributed product.
- The client may connect to the desired server on a target machine and make use of DCOM for networking. This approach may be used in conjunction with the above approach.

The use of DCOM for remote OPC Client/Server communications is the most common and obviously the approach required for cross-vendor interoperability. Consequently, there are several issues which surface in the design, development, implementation and deployment of distributed (DCOM-enabled) OPC components. This white paper will outline some of these issues that have been encountered with DCOM and OPC implementations by members of the OPC Foundation and will offer possible solutions and/or references for resolution.

Security

COM/DCOM security is by far the most misunderstood and therefore problematic issue for DCOM-enabled OPC component vendors. As DCOM uses the extensible security framework provided by Windows NT, it is essential that any discussion of DCOM security begin with an overview of this framework. The following is an excerpt from the "DCOM Technical Overview" White Paper:

DCOM can make distributed applications secure without any security-specific coding or design in either the client or the component. Just as the DCOM programming model hides a component's location, it also hides the security requirements of a component. The same (existing or off-the-shelf) binary code that works in a single-machine environment, where security may be of no concern, can be used in a distributed environment in a secure fashion.

DCOM achieves this security transparency by letting developers and administrators configure the security settings for each component. Just as the Windows NT File System lets administrators set access control lists (ACLs) for files and directories, DCOM stores Access Control Lists for components. These lists simply indicate which users or groups of users have the right to access a component of a certain class. These lists can easily be configured using the DCOM configuration tool (DCOMCNFG) or programmatically using the Windows NT registry and Win32® security functions.

Whenever a client calls a method or creates an instance of a component, DCOM obtains the client's current username associated with the current process (actually the current thread of execution). Windows NT guarantees that this user credential is authentic. DCOM then passes the username to the machine or process where the component is running. DCOM on the component's machine then validates the username again using whatever authentication mechanism is configured and checks the access control list for the component (actually for the first component run in the process containing the component. For details, see

the "DCOM Architecture" White Paper.) If the client's username is not included in this list (either directly or indirectly as a member of a group of users), DCOM simply rejects the call before the component is ever involved. This default security mechanism is completely transparent to both the client and the component and is highly optimized. It is based on the Windows NT security framework, which is probably one of the most heavily used (and optimized!) parts of the Windows NT operating system: on each and every access to a file or even to a thread-synchronization primitive like an event or semaphore, Windows NT performs an identical access check. The fact that Windows NT can still compete with and beat the performance of competing operating systems and network operating systems shows how efficient this security mechanism is.

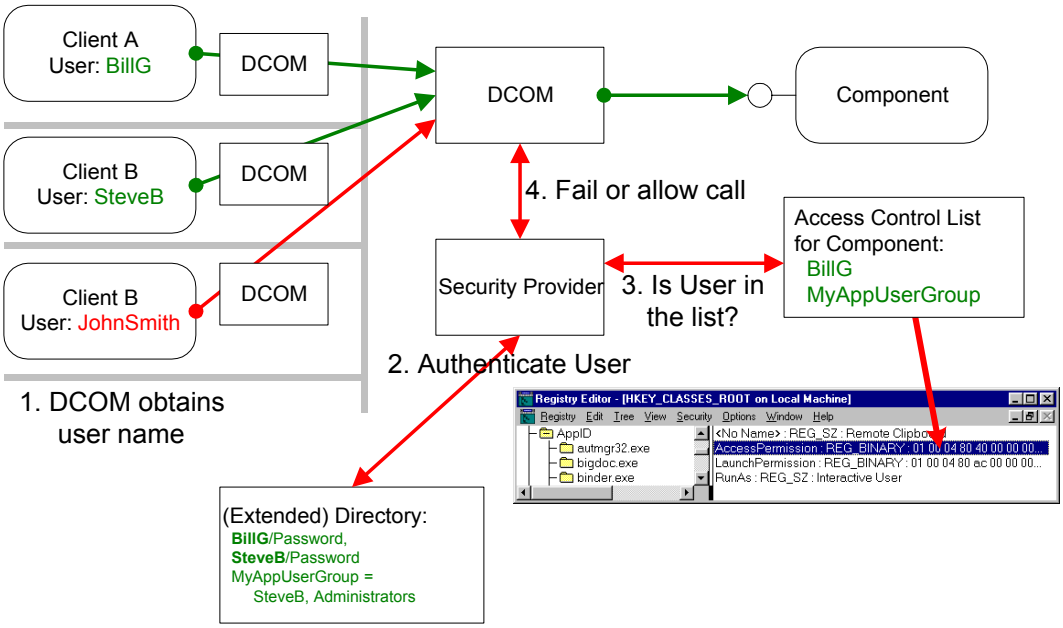

Figure 1 - Security by Configuration

There are three main issues; authentication, launch (activation) permission and access (call) permissions which all operate more or less independently of each other.

The first thing Windows NT does is to authenticate the user (as in the figure above). Whether or not this is done depends on the authentication level defined in DCOMCNFG. This level is specified by both the client and server machines: the server specifies the minimum required authentication level for incoming calls (any call that comes in below this is automatically rejected via E_ACCESSDENIED), and the client specifies it's required authentication level for each interface call. COM automatically uses the higher of the two settings. More information on these settings can be found in the HELP file for DCOMCNFG.

Once the user has been authenticated, two additional types of security are defined in DCOM: activation security (permissions) and call security (permissions).

Activation security controls which classes a client is allowed to launch and retrieve objects from, and is automatically applied by the Service Control Manager of a particular machine. Upon receipt of a request from a remote client to activate an object, the Service

Control Manager of the machine checks the request against activation setting information stored within it's registry. The HKEY_LOCAL_MACHINE\Software\Microsoft\OLE key's DefaultLaunchPermission named value sets the machine-wide default access control list (ACL) to specify who has access to classes on the system. For class-specific activation settings (which take precedence over the default setting), the HKEY_CLASSES_ROOT\APPID\{…} key's LaunchPermission named value contains data describing the class's ACL. These keys are set initially when NT is installed and can be modified by DCOMCNFG.

Call security provides the security mechanism on a per-call basis that validates inter-object communication after a connection between a client and server has been established. Call security services are divided into three categories: general functions called by both clients and servers, new interfaces on client proxies, and server-side functions and call-context interfaces. The HKEY_LOCAL_MACHINE\Software\Microsoft\OLE key's DefaultAccessPermission named value sets the machine-wide default access control list (ACL) to specify who has access to classes on the system. For class-specific activation settings (which take precedence over the default setting), the HKEY_CLASSES_ROOT\APPID\{…} key's AccessPermission named value contains data describing the class' ACL. These keys are set initially when NT is installed and can be modified by DCOMCNFG.

So again, these two sets of settings control who can Launch a COM object and also who can call methods on the object once it is launched. (In most cases you will make them the same).

See the following reference for more information on the above:

- INFO: COM Security Frequently Asked Questions – provides tips and techniques, as well as troubleshooting information, for developers of secure COM components. http://support.microsoft.com/support/kb/articles/q158/5/08.asp

Configuration

DCOM security settings may be manipulated by the DCOMCNFG utility (\WINNT\System32\dcomcnfg.exe). This utility allows the configuration of the various registry keys for both machine-wide and class-specific security settings.

Question: How do I disable DCOM security between two machines?

Background: It is often necessary to simply disable DCOM security between two machines for demo purposes or interoperability reasons.

Resolution: Use DCOMCNFG as follows:

OPC Client machine:

- Set the Default Authentication Level to (None) on the Default Properties tab.

OPC Server machine:

- Set the Default Authentication Level to (None) on the Default Properties tab.
- Add "Everyone" to both the Default Access Permissions and Default Launch Permissions Access Control Lists on the

Default Security tab.
With these settings, the domain setup (if any) as well as the IDs of the login users of each machine will be of no importance.

NOTE: if the programs involved are calling CoInitializeSecurity explicitly then there is no way to override the settings that they may have hard coded and so this procedure may not work.

Question: What is the difference in DCOM security configuration between machines located in a Workgroup (standalone) and domain environments?

Background: Certain DCOM security configurations which are suitable for OPC client/server operation on a domain (e.g. vendor development environment), may not be suitable for Workgroup or standalone (e.g. customer installations) operation.

Resolution: The issue here is authentication. In a domain environment, the domain controller (primary and/or backup) holds domain accounts that are valid on all machines that are part of the domain (as opposed to local accounts, which are valid only on the local machine). Therefore, the domain resident machine receives authentication from the domain, while the standalone machine receives authentication from it's own Security Accounts Manager (SAM). So, if authentication between the client and server is enabled, a caller's credentials must be able to be authenticated by either the domain or the local machine. For example, if a domain "OPC" is set up to run OPC components, and the machines used are logged in this domain, the desired users from the domain must be added to the components' Access Permissions and/or Launch Permission lists.

The CoInitializeSecurity() function registers security and sets the default security values for the process. For legacy applications, COM automatically calls this function with values from the registry as mentioned above. It is invoked once per process, rather than for each thread in the process. If you set registry values and then call CoInitializeSecurity(), the AppID and default registry values will be ignored, and the CoInitializeSecurity values will be used.

Question: What are the correct parameters for a CoInitializeSecurity() function?

Background: CoInitializeSecurity() may be called to programmatically set security values for a process. This function is commonly used to set the default authentication level for proxies on the process, and is the preferred method of doing so over using DCOMCNFG to set the default authentication level machine-wide. It can also set the

Access ACL for the process in case the server wants total control over it's own user list. This cannot be used to set launch permissions (this would not be useful since the call can only be made from a server which is already running).

Resolution: The following will set the default authentication level for proxies of a process to None. This would be appropriate in a low security environment. Refer to Microsoft Help for this function for additional information.

```
CoInitializeSecurity(NULL, -1, NULL, NULL,
RPC_C_AUTHN_LEVEL_NONE,
RPC_C_IMP_LEVEL_IDENTIFY, NULL, EOAC_NONE,
NULL);
```

Question: What about callbacks from the server to the client such as OnDataChange?

Background: In some cases the server will make calls back to the client. These calls are also subject to NT security. Essentially the client and server roles are reversed in this case. However in this case, only authentication is an issue, launch and access permissions are not an issue as they only apply to objects.

Resolution: In most cases (even in systems used in secure environments), the client should call CoInitializeSecurity as noted above thereby setting the authentication level of the client process to NONE. This allows the server to call back into the client. Note that only the server to which the client is connected can take advantage of this 'opening'.

Question: Can you suggest any general guidelines for setting up a secure environment?

Background: Many process control applications require a secure environment.

Resolution: This depends a great deal on the needs of the particular organization. Also there is an OPC Working group actively investigating these issues. Here are some general guidelines:

- DO use a domain.
- Set the authentication level to connect (or higher) on the server machines.
- Create a Group (in the domain) which will be allowed to launch and access the OPC objects.
- Use DCOMCNFG to include this group in the ACLs for launch and access permissions for each OPC object.
- Insure your operators are members of this group.

This should allow different users to be logged onto various client machines. It should also allow callbacks to function as the server machine can be authenticated by the client machine.

Timeouts

Question:	How can a client tell if the server is still running? And is it possible to configure the timeout interval for an OPC Client-Server method call?
Background:	Client calls to a failed server will timeout depending on the timeout provided by the transport. For example, UDP will timeout in approximately 32 seconds, with others taking up to two minutes.
Resolution:	This is a result of the transport protocol performing necessary retries, error handling, etc. Microsoft has indicated that there is no formal way of configuring this, as the required "hooks" are not present in NT 4.0 (NT 5.0 will resolve this with timeouts on the order of a few seconds). One possible workaround would be for the client implementation to set a separate timer with an appropriate timeout interval upon each method call. If the timer fires before the method call returnes, an appropriate action may be taken, such as setting affected item qualities to uncertain. **Note**: Microsoft does not recommend modifying the default IMessageFilter implementation, as has been suggested by some sources. See also 'Error handling' below.
Question:	How can my server tell if a Client is still running and does that relate to this 6 minute timeout I keep hearing about?
Background:	DCOM needs to be able to handle any combination of server, client or network crash. DCOM maintains a background 'ping' between a client and server to help with this. If a client connection is lost (e.g. the client crashed), NT will detect this within 6 minutes. At that point all of the interfaces the client had requested from the Server will be Released. This is intended to simulate a client shutting down normally.
Resolution:	Because of this, servers writers do not need to worry about client crashes. NT will essentially 'pretend' that the client shut down normally and released all interfaces.

Browsing for Remote Servers

Currently, most OPC client vendors have implemented browsing for remote OPC Servers by using the Registry API's that support access to a remote machine's registry. This method is not only a security problem, but also not recommended by Microsoft (COM server information may not be contained in the Registry in the future). An effort by the OPC Foundation is underway to resolve this issue with recommendations from Microsoft. These efforts include the use of Component Categories and/or a custom COM server which enumerates OPC Servers on the machine in which it resides and makes

these available to remote OPC clients. More to come on this…

Error Handling

When OPC Client-Server calls are remoted, all of the RPC_E_* error codes must be considered possible returns from method calls. The following may be done to indicate the need for a client to completely reconnect to a server:

```
// Check if the facility code of the returned HRESULT is of type RPC
if (FACILITY_RPC == HRESULT_FACILITY(hr))
{
    if (RPC_E_DISCONNECTED == hr)
    {
        // Need to reconnect (i.e. Call CoCreateInstance)
    }
}
```

Question:	What is the proper way for a client to determine that a server has failed and what should it do when this does happen?
Background:	The failure can occur at any time (i.e. during any method). If such a failure does occur the client will need to reconnect to the server.
Resolution:	For each interface method call, it's return value should be checked for error as indicated above. Note that Release() does not have to be called on all interface pointers in this case, as (a) the connection has been lost and such calls will not go through and (b) the server object will be garbage collected eventually (six minutes) as mentioned earlier. Note, do not use QueryInterface() as a 'ping' mechanism because a QueryInterface() for the same IID is cached and never goes remote, and proxies are not automatically updated of server status. Instead use a call such as IOPCServer->GetStatus().

Miscellaneous

Sources of information for DCOM development:

- Microsoft Mailing Lists (DCOM, ATL, etc.): http://microsoft.ease.lsoft.com/
- Microsoft Support Online (Knowledge Base): http://premium.microsoft.com/support/
- Michael Nelson's code: http://www.wam.umd.edu/~mikenel/
- Don Box's code: http://www.develop.com/dbox

References

Box, Don. Essential COM
Addison Wesley Longman, Inc., 1998

Grimes, Dr. Richard. Professional DCOM Programming

Wrox Press Ltd., 1997

"DCOM Technical Overview" White Paper
Microsoft Corporation, 1996

Revision History

Draft 1 - Monday, March 9, 1998
Revision 2 – Thursday, April 9, 1998