Enterprise Management of Windows NT Services

J. Nick Otto
notto@parikh.net
Parikh Advanced Systems

Abstract

A problem faced by NT administrators is the management of NT based services in the enterprise. In the NT environment, services have particular startup states and run with varying degrees of security. Multiple NT domains, servers, and workstations providing services all add to the difficulty of managing NT services. Un-authorized introduction of some services can cause problems to end-users and even block network access. Proactive monitoring of NT services is essential to maintaining the availability of network resources.

A system administrator must be aware of the following service related issues: 1) current status and uptime of all critical NT services; 2) services authorized to be running on the network; 3) are there any un-authorized services currently running. A system administrator must perform actions based on facts about network services. However, in order to make quality decisions, information must be collected and monitored, prior to the administrator's involvement. Each area of service management presents a unique challenge. These challenges are the motivating factor behind the tools described in this paper.

1 Introduction

This paper focuses on enterprise-level management of NT services in a large, multi-domain, NT enterprise. The tools and methodologies presented here are applicable in heterogeneous environments. They are designed to assist with the automated management of NT services.

A system leveraging Windows NT server, Visual Basic applications (with Microsoft's Active Directory Service Interfaces (ADSI)), and a Microsoft SQL server back-end was implemented to monitor and manage NT services across the enterprise.

Prior to developing the solution presented in this paper, service management solutions were investigated and tested. And, although there are many inexpensive solutions in the market for managing certain services, a custom solution was the best value. A custom solution allows more control over monitoring and negates post-implementation costs associated with scaling the application, i.e., instance licensing.

2 Systems Requirements

The NT network this solution was built for consisted of approximately 150 NT servers, 500 NT workstations, three domains, and 5,000 clients. The network is distributed over a corporate campus and a Wide Area Network.

2.1 Initial Motivation

Better automation of NT services management became a necessity when a troublesome new service was brought on-line. The service in question would randomly stop running, and require an administrator to start up the service. While working on this issue, it was determined that a tool could be written to check this service each morning and ensure it was running. After developing routines that managed the "problem" service, it was a natural progression to continue development and build a more full-featured service management solution.

2.2 Needs Assessment

The driving forces behind service management issues are NT services that stop for various reasons, fail to start, or are unavailable. Service management is different in each environment; some services are critical, others are not. Frequently services must be stopped for maintenance and should not be re-started until a later date. Based on some of these concepts, conclusions, and ideas, a production system was developed that featured the following:

- A repository of all critical NT services and associated parameters along with customizable option fields
- Administrative front-end, or console, that would allow for easy addition or removal of

monitored services, as well as stopping the monitoring process completely
- An application that would poll services and take action based on repository information
- E-mail notification, configurable per service
- Alpha paging, configurable per service
- Logging of all application functions
- A Web interface to display status, logging summaries, and other reports and statistics
- A solution built completely on Microsoft technology
- A minimal implementation cost

2.3 Requirements Analysis

The main objective of this effort was to develop a system that proactively monitors and takes reactive measures on NT services, resulting in an increase in service availability and a reduction in response times to a service outage.

During the development efforts, prototypes were compiled, posted, and run. Administrator input was gathered from the very beginning of the project and resulted in the addition of many features; most importantly the addition of the error count to limit e-mail and pages.

2.4 Functionality

Proactive service monitoring and the subsequent reactive system responses were a challenge in respect to building the application. Services can be in different states, with varying startup types. An administrator may stop a service for a specific reason and not want it started again. Critical services may require pager notification, but only during certain hours. If a service is off-line and e-mail notification is enabled, the system must be set to stop notification.

The first step in defining functional requirements was addressing the repository. The application needed to operate based on the status of a service as well as the repository information. Each service record in this database contained the following fields: Host Computer, Name, Display Name, Path, Service Account Name, Startup Type, Status, Time/Date, Pager, E-mail, and Error Count.

Services can be in one of seven states; 1) Not Running (Stopped), 2) Start Pending, 3) Stop Pending, 4) Running, 5) Continue Pending, 6) Pause Pending, and 7) Paused. A service is also configured with one of three possible startup states; 1) Automatic, 2) Manual, and 3) Disabled. Using combinations of these various service states, a set of "rules" determined the action taken by the application. The rules function based on a combination of service startup type and service status. If the service has a status of X and startup type of Y, then do Z. The basic rules represented in a case select are shown in Appendix A. Through this set of rules the system takes the appropriate action on a target service. To avoid the system "touching" a service an administrator wants off-line, the startup type can be changed to disabled, a change that can be made quickly and at the same time that the service is stopped.

The system provides for notification of actions taken. Two notifications methods were identified during the needs assessment; e-mail and alpha pager. Sending e-mail is accomplished with a simple SMTP mail routine. Alpha paging is accomplished by tying into a network accessible alpha paging solution. Functionally, the system provides both services, but one would not want to receive hundreds of e-mails if the same service was off-line or to receive excessive pages. An error count field was added to the database to compliment the fields for paging and e-mail. If the error count is above five, e-mail is no longer sent; if above two, alpha pages are no longer sent. An assumption was made that the administrator receiving the pages or e-mails will note the consecutive messages and check the service status.

The database is updated whenever a service is checked. The time and date of last service status check and the last service status are updated. Updating these fields allow for a real-time look at service statuses across the network. Along with updating the main service information, the application logs all actions taken to an error table stored in the database. Error log records contain the following fields: Error Message, Action, and Time/Date. Error log functionality is essential to the system requirements.

An administration program for the main application was developed that enumerates network resources and creates a "tree" view. Using this tool an administrator can check the available services on a server and add, remove, or edit the associated database entries. This tool also allows for "browsing" available resources. Administrators generally know their servers and which machines should be on the network. With a good naming convention one can quickly recognize servers that stand out and then check the services. An added benefit to this functionality is the ability to find rouge network servers, then take appropriate action.

Configuration data for the application is held in the database. Configuration data includes a setting for looping frequency of the service checking routine. On startup the application reads the configuration

information, then reads in all services to be checked. Once checked the application "sleeps" for a pre-set amount of time. Each iteration results in a re-reading of configuration information and of services to be checked. With this scenario, an administrator can add or remove services and expect the changes to be applied during the next run of the application.

3 Laying the Groundwork

As defined by the system needs, the service monitoring application was required to be completely built with Microsoft solutions and require minimal implementation costs. Existing development tools and back-end systems were available for this effort and helped meet the requirements. Staying within the Microsoft model allowed the system to be built and implemented on existing servers, with existing Visual Basic licenses, and the database added to an MS-SQL server that served other needs. MS-SQL server 6.5 was the database used for the implementation of this application.

3.1 Why Visual Basic?

Visual Basic 6, Enterprise, was used to develop both the administrator's console and the service monitoring application. Visual Basic provides a rapid development environment for building database driven applications. Visual Basic is easy to learn and is consistent with VBScript allowing for easy transition from application development to Active Server Page Development.

3.2 Microsoft ADSI

The Microsoft Active Directory Service Interfaces (ADSI) are a key part of this application. ADSI version 2.5 was used for this project. The ADSI controls allow objects within the NT enterprise to be connected to and manipulated. With ADSI a developer can bind to a server or the domain and control Services, Users, Printers, Shares, etc. Most core administrative tasks can be accomplished with ADSI and Visual Basic.

Another compelling argument for the use of ADSI for developing administration tools is that ADSI provides support for MS Exchange, Novell NDS, IIS, and LDAP resources. Administration tools built on this technology can bridge systems, unify administration scenarios and reduce the burden of administration in heterogeneous environments.

3.3 The role of IIS and ASP

One of the basic system requirements was to provide a status of services on the network and to provide reports. This application is completely database driven and each error log is written to the database. With all the information in the database it was a logical decision to use IIS with Active Server Pages to provide the interface for checking server status, looking up individual records and generating reports.

4 Implementation

The implementation of the service monitoring application was a straight forward process. Target domains with associated services were identified and documented. Administration groups were notified of the pending installation and trained on what the application would perform. A major function of the application is to start services that are not running. If an administrator is upgrading an application or service and the service monitoring application starts the service, there could be negative effects. Avoiding conflicts between the service monitoring application and administration teams was very important. Thus, the involvement of each administration team was essential to project success.

4.1 System Components

The system consists of four main components: 1) the service monitoring application, 2) the administration interface, 3) the database back-end, and 4) the web interface. The basic system layout is shown in Figure 1.

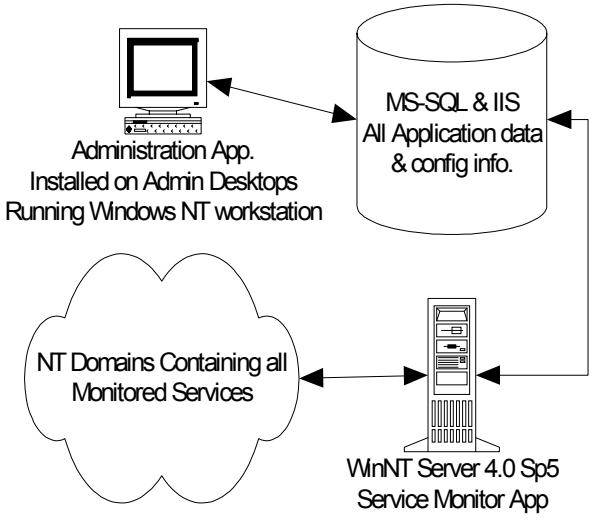

Figure 1 - Basic System Overview

4.2 Security

Since this application was built with Microsoft solutions, security is dependent upon proper settings and configurations in the Windows NT domains where the services managed reside. The services for this implementation resided in three domains, a master with two trusting sub-domains. The service monitoring application is run with domain administrator privileges. Access to the administration program and database are handled via integrated domain security with the MS-SQL server.

4.3 Propagating the database

There were two options available for database propagation: 1) enumerate network resources and add services based on a set of criteria, or 2) selectively add services to the database with the administration utility. Option 2 was chosen for database propagation so that subjective reasoning could be applied to which services were added. Only critical services were added to the database, there was no need to fill the database with extraneous services. If an automatic procedure was used to fill the database, services from all enumerated servers would be in the database. There was no desire to add "test" or other non-critical servers to the program. Also, for each service there is the option for e-mail and/or pager notification upon taking an action on the service. The decision for these settings is completely subjective and up to the administrator.

The process to add all the desired services to the database was approximately a half-day effort. Each server was investigated with the administration program and a decision made on which services should be added and the associated options. After implementation, new services can be added to the database as part of the process of adding a new server to the network.

4.4 Deployment

Deployment of the application took place as the program was being developed. One of the basic system needs was to have a solution that would be installed on currently available hardware. The database was added to an existing server and the application was installed on an administration staff utility server. There was no "roll-out" in the traditional sense. The application did require installation of ADSI on the server that hosted the application. There is no client component to this application. All work is performed by the service monitoring application with all communication and polling originating with the application. This situation is helpful when monitoring services in a firewall DMZ, as all connections originate from inside the network from the server hosting the service monitoring application (although, this was not done for this implementation).

4.5 Testing

Testing consisted of manually changing services to varying states and watching to see if the application behaved as predicted. During this process it was discovered that "hung" services did not always return an expected response to the application, sometimes the service would return nothing. This situation, and the case of a server being off the network, was handled by the error handling routines in the program. If the service didn't return a status, or was not available, a status of "un-reachable" was added to the database and the appropriate notification carried out.

The application proved to be very reliable and accurate in determining what action to take. Many of the monitored services were over a WAN, with some very slow links, and the application handled the slow connections without problems. Sleep routines were built in to account for start-up delays when the application started a service. Originally the application was to issue the start command, then check on the next iteration of the program. However, this makes it more difficult to track what action the program has previously taken. To solve the problem, the program re-checks any service it takes action on, prior to moving to the next service.

4.6 Support Structure and Monitoring

Prior to implementation, service monitoring and management was a reactive process. A service would become un-available and remain so until reported by a user or noticed by an administrator. Service management under the old scenario resided with first level administrators who would typically just re-start the service, (or re-boot the server), and then report the incident to the appropriate group. The service monitoring application changes this scenario in that the re-starting of a service, while still a reactive solution, takes action prior to user notification. This results in minimizing the time a service is un-available.

E-mail and pager notifications are handled by sending the e-mail messages to a mail group that includes all administrators. The "on-call" administrator can check the messages and take action; while other administrators can just delete the messages. This setup avoids the issue of making weekly changes to the e-mail configuration options. It also allows for coverage

when the "on-call" administrator is busy and the back-up "on-call" would be seeing the messages. The alpha-pages are sent to an "on-call" pager that is passed around the support group. Pages from the application are treated in the same manner as any other support call.

Monitoring of log files and application activity allows for service monitoring to become a more proactive situation. A given service may stop for an unknown reason on a regular basis. Administrators may be re-starting this service but not readily noticing the trend. Analysis of the log files and system activity allows for trend recognition and association of application activity with other processes. Thus, a problem service may not be the problem at all, just the end result of a different issue. Analysis of log files for this implementation resulted in the identification of several trends that were corrected shortly after implementation.

5 Lessons Learned – Success?

Post-implementation the question was asked, "Were the needs met and does the system work?" The short answer is yes. The suite of Microsoft products chosen proved to be reliable and extensible. Concerns about programmatically managing services were raised and investigated. This simple automation of existing technology proved to be essential for administering large environments where consolidation of effort is important. The application functioned as expected and resulted in identifying troublesome services and root causes. The web interface proved to be a handy addition as this interface can easily be sorted by category. An administrator can check the current status of all instances of a given service, or can just check for any service in the system not currently running.

5.1 Problems and Bugs

The only major problem encountered revolved around network outages. If a WAN link or network segment went down that contains several services, the administration team is hit with many e-mails and pages. This problem is a design issue and can be changed. The work around used for this problem was to stop service monitoring in the event of a network outage. The network in question was very robust and the situation of having to stop service monitoring rarely occurred.

5.2 Extensibility and Future improvements

The main purpose of this application was to monitor and report on services existing within a WinNT domain or domains. However, building the application with ADSI allows for features to be added that can change the application from a service monitoring tool to a service administration tool.

The application can easily be modified to allow for making changes to the service properties, both the administrator's console and web interface can be modified in this way. This would allow an administrator checking service status the ability to change the status with the tool. The startup type and associated account information could then also be modified.

Additionally, ADSI provides support for many other environments and products. Using this support the service monitoring application can be extended to manage resources other than NT services. In a nutshell, the framework chosen for this application allows for extending the application to many aspects of systems administration. Using Visual Basic as the development tool, also allows for taking advantage of the Windows API to add features not supported directly by ADSI.

6 Summary

By leveraging existing resources and the power of Visual Basic with ADSI, a working service-management solution was implemented. Monthly reports show when and where there are issues with key network resources. Troublesome services (those services that tend to stop on their own), can now be re-started expeditiously, reducing downtime and calls to the help desk, and identifying root causes. The logging and tracking functionality provides the ability to quickly recognize trends and problem areas. The implementation resulted in an increased level-of-service and a corresponding decrease in administrative response time.

7 References

15 Seconds Web Page, http://www.15seconds.com.

Eck, Thomas, Windows NT/2000 ADSI Scripting for System Administration, MTP, 2000

Hahn, Steven, ADSI ASP Programmer's Reference, Wrox Press, 1998.

Microsoft Active Directory Services Interfaces Overview pages, http://www.microsoft.com/windows2000/library/howitworks/activedirectory/adsilinks.asp

Appendix A – Code Examples

The code examples show how the decision structure for the service monitoring application works. The GetServiceStatus function is called by the main program to check the current status and startup state of a specified service. If the service is running or paused, the function returns to the main routine. Any state other than paused or running results in the StartService function being called. A rather verbose case select is used during the StartService function to determine if the service starts and if a page should be sent. This code is dependent on several other routines and is not intended to be fully functional.

```
' This function shows the decision structure to decide if the application should take
' an action on a service.  Service to be checked is passed from main routine.
' Return values for service status and start type are defined with constants.
' Actual values can be found in the ADSI documentation.

Function GetServiceStatus(sServiceName As String) As String

Dim ServiceStats As IADsServiceOperations
Dim ServiceInfo As IADsService
Dim Status As Integer

On Error GoTo ErrHandler

Set ServiceStats = GetObject(sServiceName)

Status = ServiceStats.Status

' if the status is running or paused, leave, we're done

If Status = S_Running Then
    GetServiceStatus = "Running"
    iCurrErrCount = "0"
ElseIf Status = S_Paused Then
    GetServiceStatus = "Paused"
    iCurrErrCount = "0"
Else
    Set ServiceInfo = GetObject(sServiceName)
    ' if service is not disabled, pass the service to the startservice function,
    ' if not end routine.
    If Not ServiceInfo.StartType = S_Disabled Then
        GetServiceStatus = StartService(sServiceName)
    Else
        GetServiceStatus = "Stopped(Disabled)"
        iCurrErrCount = "0"
        AddErrLogEntry sServiceName, "Checked service has a start value of Disabled", _
                                     "Re-Start not attempted"
    End If
End If

Set ServiceStats = Nothing
Set ServiceInfo = Nothing

Exit Function

ErrHandler:
    GetServiceStatus = "UnReachable"
    AddErrLogEntry sServiceName, "Service Unavailable or Server off Network", "None"

    iCurrErrCount = iCurrErrCount + 1

    Err.Clear

    Set ServiceStats = Nothing
    Set ServiceInfo = Nothing

End Function
```

```
' this function is used to start services and then pass the results back to the
' getservice status function.  Here again constants are used to identify the return
' value of service status.

Function StartService(sServiceName As String) As String

Dim ServiceOp As IADsServiceOperations
Dim result As String

Set ServiceOp = GetObject(sServiceName)

' We know the service is stopped, so let's start it
ServiceOp.Start

MeSleep (5)  ' wait for service to start

' This case select is used to determine what the application should do
' after the first service startup has been attempted.

Select Case ServiceOp.Status
    Case Is = S_Not_Running
        StartService = "Stopped(Re-Start Failed)"
        AddErrLogEntry sServiceName, "Service not running", "Re-Start attempt failed"
        If iCurrErrCount = "0" And sIsPageable = "Yes" Then
            Send_Page sServiceName & ": Service not running.  Re-Start attempt failed."
        End If
        iCurrErrCount = iCurrErrCount + 1
    Case Is = S_Start_Pending
        MeSleep (5)
        If ServiceOp.Status = S_Running Then
            StartService = "Running(Re-Started)"
            AddErrLogEntry sServiceName, "Service not running", "Re-Started"
            iCurrErrCount = "0"
        Else
            StartService = "Stopped(Re-Start Failed)"
            AddErrLogEntry sServiceName, "Service not running", "Re-Start attempt failed"
            If iCurrErrCount = "0" And sIsPageable = "Yes" Then
                Send_Page sServiceName & ": Service not running.  Re-Start failed."
            End If
            iCurrErrCount = iCurrErrCount + 1
        End If
    Case Is = S_Stop_Pending
        MeSleep (5)
        If ServiceOp.Status = S_Running Then
            StartService = "Running(Re-Started)"
            AddErrLogEntry sServiceName, "Service not running", "Re-Started"
            iCurrErrCount = "0"
        Else
            StartService = "Stopped(Re-Start Failed)"
            AddErrLogEntry sServiceName, "Service not running", "Re-Start attempt failed"
            If iCurrErrCount = "0" And sIsPageable = "Yes" Then
                Send_Page sServiceName & ": Service not running.  Re-Start failed."
            End If
            iCurrErrCount = iCurrErrCount + 1
        End If
    Case Is = S_Running
        StartService = "Running(Re-Started)"
        AddErrLogEntry sServiceName, "Service not running", "Re-Started"
        iCurrErrCount = "0"
    Case Is = S_Continue_Pending
        MeSleep (5)
        If ServiceOp.Status = S_Running Then
            StartService = "Running(Re-Started)"
            AddErrLogEntry sServiceName, "Service not running", "Re-Started"
            iCurrErrCount = "0"
        Else
            StartService = "Stopped(Re-Start Failed)"
            AddErrLogEntry sServiceName, "Service not running", "Re-Start attempt failed"
            If iCurrErrCount = "0" And sIsPageable = "Yes" Then
                Send_Page sServiceName & ": Service not running.  Re-Start failed."
            End If
```

```
            iCurrErrCount = iCurrErrCount + 1
        End If
    Case Is = S_Pause_Pending
        MeSleep (5)
        If ServiceOp.Status = S_Running Then
            StartService = "Running(Re-Started)"
            AddErrLogEntry sServiceName, "Service not running", "Re-Started"
            iCurrErrCount = "0"
        Else
            StartService = "Stopped(Re-Start Failed)"
            AddErrLogEntry sServiceName, "Service not running", "Re-Start attempt failed"
            If iCurrErrCount = "0" And sIsPageable = "Yes" Then
                Send_Page sServiceName & ": Service not running.  Re-Start failed."
            End If
            iCurrErrCount = iCurrErrCount + 1
        End If
    Case Is = S_Paused
        MeSleep (5)
        If ServiceOp.Status = S_Running Then
            StartService = "Running(Re-Started)"
            AddErrLogEntry sServiceName, "Service not running", "Re-Started"
            iCurrErrCount = "0"
        Else
            StartService = "Stopped(Re-Start Failed)"
            AddErrLogEntry sServiceName, "Service not running", "Re-Start attempt failed"
            If iCurrErrCount = "0" And sIsPageable = "Yes" Then
                Send_Page sServiceName & ": Service not running.  Re-Start failed."
            End If
            iCurrErrCount = iCurrErrCount + 1
        End If
End Select

Set ServiceOp = Nothing

Exit Function
```