Es

The Monad Manifesto (Annotated)

Jeffrey Snover
Original Author

The Monad Manifesto: Annotated (Spanish)

The DevOps Collective, Inc.

Este libro está a la venta en
http://leanpub.com/monad-manifesto-annotated-spanish

Esta versión se publicó en 2018-10-28

Este es un libro de Leanpub. Leanpub anima a los autores y publicadoras con el proceso de publicación. Lean Publishing es el acto de publicar un libro en progreso usando herramientas sencillas y muchas iteraciones para obtener feedback del lector hasta conseguir tener el libro adecuado.

© 2018 The DevOps Collective, Inc.

También por The DevOps Collective, Inc.

Creating HTML Reports in Windows PowerShell

A Unix Person's Guide to PowerShell

The Big Book of PowerShell Error Handling

DevOps: The Ops Perspective

Ditch Excel: Making Historical and Trend Reports in PowerShell

Secrets of PowerShell Remoting

The Big Book of PowerShell Gotchas

The Monad Manifesto, Annotated

Why PowerShell?

Windows PowerShell Networking Guide

The PowerShell + DevOps Global Summit Manual for Summiteers

Why PowerShell? (Spanish)

Secrets of PowerShell Remoting (Spanish)

DevOps: The Ops Perspective (Spanish)

Creating HTML Reports in PowerShell (Spanish)

The Big Book of PowerShell Gotchas (Spanish)

The Big Book of PowerShell Error Handling (Spanish)

DevOps: WTF?

PowerShell.org: History of a Community

Índice general

The Monad Manifesto - Annotated

by Jeffrey Snover as annotated by the PowerShell Community

Este proyecto está destinado a preservar The Monad Manifesto[1], un documento escrito por el inventor de Microsoft Windows PowerShell Jeffrey Snover[2] en Microsoft en 2002. La idea de este proyecto fue del autor de Pluralsight Tim Warner[3], con las anotaciones iniciales que hicieroen Tim y el Microsoft MVP Don Jones[4].

El Manifiesto original era un documento prospectivo, anterior a la publicación pública de PowerShell por alrededor de 4 años. En los años transcurridos desde el lanzamiento de PowerShell 2006[5], el producto ha evolucionado sustancialmente, pero siempre alrededor de los conceptos descritos en el Manifiesto.

Consideramos que no sólo es importante conservar el documento con fines históricos, sino también anotar y ampliar los diversos conceptos que introduce. Intentaremos vincular las referencias de las tecnologías reales que el Manifiesto predijo y proporcionar explicaciones contextuales en torno a algunas de las directivas del Manifiesto.

Encontrará [^1] notas de pie de página en el texto. Éstas son una característica de MultiMarkdown[6] que no son compatibles con

[1] http://www.jsnover.com/blog/2011/10/01/monad-manifesto/

[2] https://social.technet.microsoft.com/profile/Jeffrey%20Snover%20Windows%20Server

[3] http://www.pluralsight.com/author/tim-warner

[4] https://twitter.com/concentrateddon

[5] http://blogs.msdn.com/b/powershell/archive/2006/11/14/windows-powershell-1-0-released.aspx

[6] http://fletcherpenney.net/multimarkdown/

nuestra plataforma de publicación, pero están destinadas a vincular a las notas de pie de página correspondientes en la parte inferior de la página. En algunos casos, estas son las notas originales de Jeffrey marcadas como "ORIGINAL" para separarlas de las notas a pie de página que nosotros hemos añadido.

Esta guía se publica bajo la licencia Creative Commons Attribution-NoDerivs 3.0 Unported. Los autores le animan a redistribuir este archivo lo más ampliamente posible, pero le solicitan que no modifique el documento original.

¿Ha sido útil este libro? El (los) autor (es) le pide (n) que haga una donación deducible de impuestos (en los EE.UU., consulte sus leyes si vive en otro lugar) de cualquier cantidad a The DevOps Collective[7] para apoyar su trabajo.

** Revise las actualizaciones! ** Nuestros ebooks se actualizan a menudo con contenido nuevo y corregido. Los hacemos disponibles de tres maneras:

- Nuestra rama principal GitHub organization[8], con un repositorio para cada libro. Visite https://github.com/devops-collective-inc/
- Nuestra GitBook page[9], donde puede navegar por los libros en línea, o descargarlos en formato PDF, EPUB o MOBI. Utilizando el lector en línea, puede saltar a capítulos específicos. Visite https://www.gitbook.com/@devopscollective
- En LeanPub[10], donde se pueden descargar como PDF, EPUB, o MOBI (login requerido), y "comprar" los libros haciendo una donación a DevOps. También puede elegir recibir notificaciones de actualizaciones. Visite https://leanpub.com/u/devopscollective

[7] https://devopscollective.org/donate
[8] https://github.com/devops-collective-inc
[9] https://www.gitbook.com/@devopscollective
[10] https://leanpub.com/u/devopscollective

GitBook y LeanPub generan la salida del formato PDF ligeramente diferente, por lo que puede elegir el que prefiera. LeanPub también le puede notificar cada vez que liberamos alguna actualización. Nuestro repositorio de GitHub es el principal; los repositorios en otros sitios suelen ser sólo espejos utilizados para el proceso de publicación. GitBook normalmente contendrá nuestra última versión, incluyendo algunos bits no terminados; LeanPub siempre contiene la más reciente "publicación liberada" de cualquier libro.

Capítulo 1 - ¿Qué es Monad?

Monad [11] [12] es la próxima generación de plataformas para la automatización administrativa. Monad resuelve los problemas de gestión tradicionales aprovechando la Plataforma .NET[13]. Desde el primer prototipo (limitado), se pueden resaltar beneficios significativos para desarrolladores, testers, usuarios avanzados y administradores. Monad aprovecha [^ 1-6] el .NET Common Runtime[14] Runtime para proporcionar un potente, consistente, intuitivo, extensible y útil conjunto de herramientas que reducen los costos de administración y hacen que la vida de los no programadores sea mucho más sencilla.

Monad consta de:

1. Monad Automation Model (MAM)[15]: Un modelo de automatización basado en clases .NET[16], métodos y atributos para

[11](ORIGINAL) Este no es un documento técnico de Windows PowerShell ni es una descripción precisa de cómo funciona V1.0. Esta es una versión del original Manifiesto de Monad que articuló la visión a largo plazo y comenzó el esfuerzo de desarrollo que se convirtió en PowerShell. Muchos de los elementos descritos en este documento han sido liberados y otros han proporcionado una buena hoja de ruta para el futuro. El documento se ha actualizado para su publicación. La información confidencial ha sido eliminada y los ejemplos se actualizan para reflejar la sintaxis actual.

[12](ORIGINAL) Monad es el término de Leibniz's utilizado para describir una unidad fundamental a la que luego se agregan componentes para implementar un propósito. En esta filosofía, todo es una composición de Monads. Esto captura lo que queremos lograr con una gestión compuesta. Más información sobre Monad se puede encontrar en: http://www.wise.virginia.edu/philosophy/phil206/Leibniz.html

[13]http://bit.ly/1PAsRao

[14]http://bit.ly/1Q0TrV3

[15]

[16]http://bit.ly/1R9oPTO

producir Cmdlets[17].[18]

2. Monad Shell (MSH)[19]: Un entorno de ejecución de scripts basado en .NET para exponer los Cmdlets como herramientas de línea de comandos de API[20] y un shell de línea de comandos programable e interactivo.
3. Monad Management Models (MMM)[21]: El conjunto con las clases de base de código administrado (o interfaces) para implementar escenarios de administración específicos y herramientas administrativas in-the-box para ejecutar esos escenarios.
4. Monad Remote Scripting (MRS)[22]: Conjunto de componentes basados en Web Services[23] que permiten ejecutar secuencias de comandos remotamente en muchas máquinas [24].
5. Monad Management Console (MMC)[25]: Un modelo basado en .NET y un conjunto de servicios para la creación de GUIs de administración sobre MSH[26] exponiendo todas las interacciones de GUI como secuencias de comandos visibles por el usuario [27].

[17] https://msdn.microsoft.com/en-us/library/ms714395(v=vs.85).aspx

[18] La versión 1 de PowerShell se liberó en 2006 y proporcionó la implementación de cmdlets. Los cmdlets de hoy se escriben en lenguajes .NET y consisten en una sola clase por cada cmdlet. PowerShell proporciona una clase base que hace mucho del trabajo pesado. Los desarrolladores definen las propiedades de la clase que se convierten en parámetros y reemplazan métodos específicos para participar en el ciclo de vida de la canalización en el pipeline. Los cmdlets, junto con el entorno general, fueron el primero de los cuatro puntos de visión principales que se proponen en el Manifiesto.

[19]

[20] https://msdn.microsoft.com/en-us/library/ms123401.aspx

[21]

[22]

[23] https://msdn.microsoft.com/en-us/library/ms950421.aspx

[24] Remoting fue introducido en PowerShell versión 2, que se liberó con Windows Vista y Windows Server 2008. Remoting es el segundo de los cuatro puntos de visión principales propuestos en el Manifiesto.

[25]

[26] https://technet.microsoft.com/en-us/magazine/2005.11.scripting.aspx

[27] Aunque nunca se expuso como una MMC per se, el motor de PowerShell se implementó como una clase .NET. Cualquier aplicación .NET puede instanciar el motor, ejecutar comandos y traducir la salida a una pantalla GUI. Exchange Server 2007 fue el primer producto que lo hizo y sigue siendo uno de los mejores ejemplos del "enfoque completo de PowerShell" para la administración.

Este white paper[28] presenta el enfoque tradicional de la automatización administrativa, sus fortalezas y deficiencias. A continuación, se presenta una visión general de los principales componentes de Monad. Un conjunto de propuestas de valor[29] se articula entonces para las audiencias objetivo de Monad.

Notas:

[28] https://en.wikipedia.org/wiki/White_paper
[29] https://en.wikipedia.org/wiki/Value_proposition

Capítulo 2 – El Problema

Windows tiene herramientas administrativas GUI simples para usuarios básicos (Panel de control, MMC, etc.). Windows también tiene un rico conjunto de lenguajes, APIs [30] y modelos de objetos para programadores de sistemas avanzados (C[31], C++[32], C#[33], WMI[34], Win32[35], .NET, etc.). Lo que falta, son herramientas "compuestas" vitales orientadas al administrador para escribir comandos y automatizar la gestión. El centro de todo esta normalmente regido por lenguajes de scripting.

Nuestras soluciones de secuencias de comandos actuales (WSH[36], VB[37]) se centran en el extremo superior del mundo de secuencias de comandos que gestionan la plataforma utilizando abstracciones de muy bajo nivel, como modelos de objetos complejos, esquemas y API [38]. Esto puede resultar algo extraño para gran parte de la comunidad de administradores. El scripting de administración

[30] De hecho, las API son el diferenciador principal entre los sistemas Windows y Linux/UNIX. En Linux/UNIX, todo se parece esencialmente a una carpeta o un archivo, y casi todos los bits de configuración se encuentran en un archivo de texto de estructura libre. La automatización de la administración en ese entorno es fácil, ya que sólo tiene una API: archivos de texto. Windows es más difícil porque para hacer algo, tiene que aprender alguna API - y todas las API son diferentes. Saber cómo agregar un usuario a Active Directory no le ayuda a crear un sitio en SharePoint: todas son API diferentes.

[31] http://bit.ly/1SmIDVh

[32] http://bit.ly/1HmcYe5

[33] http://bit.ly/1EngdQ6

[34] http://bit.ly/1ekpnrY

[35] http://bit.ly/1IORfB2

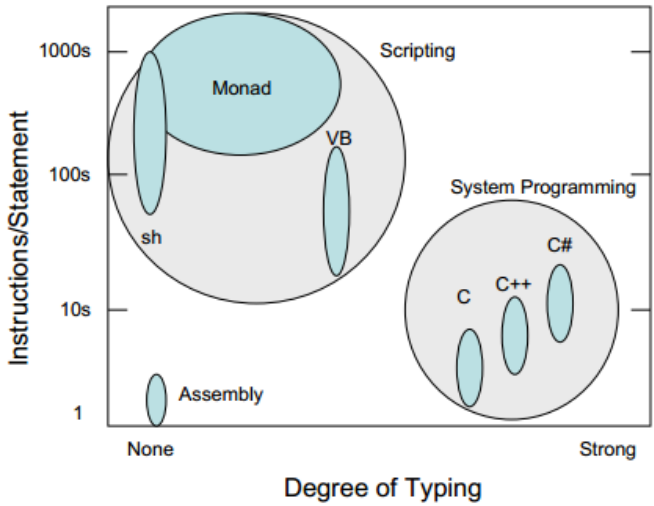
[36] http://bit.ly/1ekpvra

[37] http://bit.ly/1Q0VwjT

[38] En otras palabras, no se alcanza el objetivo, porque VBScript es básicamente una forma simplificada de tratar con las API que estaban destinadas a los desarrolladores. VBScript también asume que los equipos de producto han creado API dedicadas, compatibles con VBScript, lo que la mayoría no hizo. Conseguir algo con VBScript era a menudo complicado, y siempre a punta de prueba-error.

debería fluir desde línea de comandos [39], debería ser pequeño, simple, incremental y tratar con niveles de abstracción muy altos.

John Ousterhout[40] describió la distinción entre scripting y programación de sistemas en su artículo Scripting: Higher Level Programming for the 21st Century[41].



Degree of Typing

Ousterhout[42] postula que las secuencias de comandos deben permitir "juntar" aplicaciones, una abstracción de nivel superior a la programación de sistemas, lo que permitía un desarrollo de aplicaciones aún más rápido que con los actuales lenguajes de programación. El argumento fundamental es que debemos conti-

[39](ORIGINAL) El scripting administrativo es a menudo la progresión de scripts ad hoc a operaciones automatizadas. Los administradores advierten que escriben los mismos comandos una y otra vez así que mejor construyen una secuencia de comandos. Ellos se percatan que sus secuencias de comandos siempre contienen muchas de las mismas cosas por lo que producen subrutinas parametrizadas y avanzan desde allí.

[40]http://web.stanford.edu/~ouster/cgi-bin/home.php

[41]http://web.stanford.edu/~ouster/cgi-bin/papers/scripting.pdf

[42]http://web.stanford.edu/~ouster/cgi-bin/home.php

nuar por el camino de la Ley de Moore[43] para llevar el desarrollo a niveles más altos de abstracción a través del scripting. Para habilitar la automatización de la administración en el mainstream, los administradores necesitan un shell completo, con scripts y utilitarios, y las GUIs administrativas[44] necesitan estar superpuestas a esta infraestructura [^2-4]. Esto permitiría una formación eficiente de los administradores en la automatización desde la línea de comandos y garantizaría capacidades administrativas completas así como economías de escala en un modelo de automatización al que llama admin-composable. ___

Notas

[^2-4] Se tenía entonces una fuerte dependencia de las capas de GUIs. Eso explica un poco la ausencia de algunas GUIs administrativas en Linux/UNIX para algunas tareas. Su ausencia obliga a asegurarse que todo se puede hacer desde la línea de comandos. La GUI no se convierte en una clase especial de ciudadano que posee poderes especiales y únicos. Sólo es otro consumidor de la línea de comandos. La línea de comandos, a su vez, puede ser consumida más fácilmente por otros públicos diferentes a una GUI.

[43] http://www.mooreslaw.org
[44] https://notgartner.wordpress.com/2008/02/23/how-to-host-the-powershell-runtime/

Capítulo 3 - El enfoque tradicional de la automatización administrativa

El modelo tradicional [45] para la automatización administrativa es potente y exitoso. Consiste en:

1. Un shell de programación (por ejemplo, sh, csh, ksh, bash) [46]
2. Un conjunto de comandos administrativos (por ejemplo, ifconfig, ps, chmod, kill)
3. Un conjunto de utilitarios para manipulación de texto (por ejemplo, awk, grep, sed).
4. GUIs administrativas superpuestas a los comandos y los utilitarios

La filosofía de este modelo es que cada ejecutable debe hacer un pequeño conjunto de funciones, para que las funciones más complejas sean compuestas a través del uso de pipelining o una secuencia de ejecutables que se llaman unos a otros. Este modelo ha sido muy exitoso a pesar de tener algunos inconvenientes. Tras la inspección, lo que es ampliamente considerado como un bastión

[45] Tradicional en el mundo Linux/Unix, pero no en Windows. Este es, de hecho, el cambio que Snover proponía: hacer que la administración administrativa funcione más como en Unix, ya que Unix es un modelo probado de éxito desde hace décadas. Probablemente ayudo que Snover proviniera de Digital Computer, una compañía con bastante familiaridad en variantes de Unix y sistemas operativos similares.

[46] Estos ejemplos enfatizan la influencia que el mainframe y UNIX tenían en las opciones de diseño de Snover.

UNIX es de hecho una implementación defectuosa de este modelo [47].

Cuando retrocede y examina lo que realmente sucede cuando alguien usa un comando pipelinado (si se me permite inventar esta palabra) como en "$ a | b | c", se concluye que el primer comando, es decir, "a" no logró lo que el administrador quería hacer. Si lo hubiera hecho, el administrador sólo tendría que haber escrito "a" y listo. Entonces, la pregunta es ¿por qué "a" no hizo lo que el administrador quería? La respuesta es que en este modelo tradicional, los ejecutables autónomos unen firmemente tres operaciones: 1) obtener objetos; 2) procesamiento de objetos; 3) salida de resultados como texto [48]. Una de esas operaciones no hace lo que el administrador necesita, así que el resto de la tubería es un intento de corregir eso.

Debido a que el ejecutable genera texto, los elementos descendentes deben utilizar utilidades de manipulación de texto para intentar volver a los objetos originales y realizar trabajo faltante. Si bien el modelo básico es extremadamente poderoso, su defecto intrínseco es la estrecha vinculación de estas operaciones y el uso de texto no estructurado para la integración [49]. Esto requiere utilitarios para la

[47] Las personas que ven PowerShell como "linux-ification" de Windows deben tener en cuenta que Snover no estaba enamorado del modelo de línea de comandos Unix. Él sentía que era inconsistente y que le faltaba una mejor semántica. De muchas maneras, PowerShell fue el primer "segundo vencedor" en el modelo de línea de comandos de Unix, tomando sus puntos fuertes, pero reconsiderando lo que se había convertido en debilidades algo obvias.

[48] El resultado práctico de esto es que las herramientas - cmdlets, en el mundo de PowerShell - deben hacer una cosa, y sólo una cosa. Obtener objetos, procesar objetos o formatear objetos desde texto. Elija sólo una cosa y haga sólo eso. Si hace más de una cosa, comienza a crear una herramienta monolítica que es menos fácil de reutilizar. Este concepto de una sola cosa se ha convertido en el fundamento de las mejores prácticas en la comunidad de PowerShell, especialmente en torno a la creación de herramientas.

[49] Hay un punto enorme aquí que a menudo se pierde. Cuando se escribe una herramienta que produce texto, las herramientas descendentes tienen que saber cómo procesar ese texto en el formato exacto que se produjo. Sus datos no están estructurados. Si cambia la salida de su herramienta, todo lo que se utiliza para trabajar con ella tendrá que cambiar. La orientación de objetos, es decir, presentar los datos en una estructura estandarizada que podría ser consumida por cualquier cosa que entienda "objetos", fue una de las mayores diferencias entre PowerShell y lo que había antes. Gran parte del tiempo de un administrador de Linux se gasta en el ciclo grep/sed/awk, ya que tienen que analizar el texto para que la próxima herramienta tenga datos con los que trabajar. PowerShell casi que elimina ese trabajo por completo.

manipulación de texto torpes, con pérdidas e imprecisiones.

El modelo tradicional refleja el estado de la tecnología que estaba disponible en el momento en que surgió. .NET proporciona [50] un nuevo conjunto de capacidades y abre la posibilidad de nuevos enfoques. Estos nuevos enfoques nos permiten sustituir el modelo tradicional por uno decisivamente superior. Ese modelo es lo que llamamos Monad

---

Notas

[50] De manera realista, COM podría haber proporcionado las mismas capacidades ya que estaba orientada a objetos. Sin embargo, en el momento en que se escribió el manifiesto, COM fue "acabado" y Microsoft se había trasladado a .NET

Capítulo 4 - Nuevos enfoques

Monad adopta nuevos enfoques a los problemas de 1) construcción de comandos, 2) composición de soluciones 3) modelos de gestión y 4) GUI de gestión. La arquitectura de Monad proviene de las siguientes observaciones:

1. La mayoría de las soluciones son desarrolladas "in house" y compuestas por comandos existentes por los administradores.
2. La mayoría de las soluciones se centran en la automatización de la gestión o la provisión de correcciones ad hoc.
3. La mayoría de los administradores no son programadores "natos". O bien no tienen el deseo, la habilidad o (más a menudo), el tiempo para hacer una programación sofisticada.
4. La mayoría de los desarrolladores de aplicaciones no harán que su código sea manejable a menos que haya un beneficio inmediato y sustancial para el usuario [51]

[51] Lo que significa que la mayoría de los desarrolladores no implementarán interfaces que los administradores puedan usar para administrar la aplicación. En el mejor de los casos, un desarrollador "perezoso" podría simplemente poner toda su información de configuración en un archivo de texto y llamarla "manejable". Irónicamente, eso es esencialmente como Unix se construyó desde cero, y *es manejable*, porque no es tan fácil como modificar un archivo de texto, especialmente si está estructurado (como en JSON o XML).

4.1 - Un nuevo enfoque para construir comandos

El enfoque tradicional de la construcción de comandos es ineficiente. Gran parte del esfuerzo se dedica a reescribir las mismas funciones una y otra vez por diferentes personas de diferentes maneras. Todos para:

- Analizar, validar y codificar la entrada de usuario.
- Documentar su uso.
- Dejar registro de actividades.
- Formatear datos, resultados de salida e informes de errores.
- Operar en nodos remotos o conjuntos de nodos remotos.

Sin embargo, a pesar de toda esta coincidencia, la mayoría de las plataformas [^4-1] [52] proporcionan poco o ningún apoyo para hacer estas actividades de manera coherente. El resultado es que los comandos de hoy en día son ineficientes para desarrollar e inconsistentes en su forma de usar [53].

Monad adopta un enfoque diferente que proporciona a los desarrolladores el máximo aprovechamiento y la máxima consistencia para los usuarios finales, mediante la definición de un modelo de automatización para aplicaciones que afecta a las funciones comunes para que puedan implementarse una vez en un entorno de ejecución común [54]. Los desarrolladores ya no producen ejecutables autónomos. En su lugar, escriben piezas de código muy enfocadas como clases .NET (Cmdlets) que luego se exponen como API,

[52] ORIGINAL: VMS DCL y AS400's CL son las excepciones a esto. Proporcionan un analizador de comandos común para que los comandos que se usan tengan un alto grado de consistencia sintáctica.

[53] Es por eso que los desarrolladores odian hacerlos y los administradores odian usarlos.

[54] ORIGINAL: Existe una maravillosa sinergia entre el deseo del programador de minimizar la cantidad de código que escribe para la administración y los clientes que desean tener una experiencia de administración consistente.

comandos e interfaces gráficas. Las funciones comunes se implementan y prueban una vez y proporcionan un conjunto único de semántica, así como un conjunto coherente y uniforme de mensajes de error. [55]

4.2 - Un nuevo enfoque para componer soluciones

El enfoque tradicional para componer soluciones es difícil y frágil. Utiliza "pipelines" para realizar análisis basado en oraciones de flujos de texto [56]. Estos mecanismos son incómodos, inconsistentes e imprecisos. Los administradores pasan la mayor parte de su tiempo buscando mecanismos para resolver problemas, en lugar de resolver dichos problemas. Monad tiene un enfoque diferente que proporciona un motor de ejecución de secuencias de comandos preciso y potente para crear tuberías (pipelines) de objetos .NET. En lugar de canalizar texto no estructurado, canalizamos objetos .NET [57]. Esto permite que los componentes de la canalización (pipelines) operen a bajo nivel directamente sobre los objetos y sus propiedades utilizando las API .NET Reflection[58]. (Las API de Reflection permiten encontrar el tipo de un objeto, las propiedades

[55]Este es el modelo adoptado por PowerShell. Los cmdlets son instancias de una clase, que heredan de una única base. Esa clase proporciona una tonelada de funcionalidad común, de modo que el código real en un cmdlet está alrededor del 99% centrado en lo que sea que el cmdlet esté haciendo. El desarrollador del cmdlet no se centra en analizar los argumentos de la línea de comandos, validar los elementos obligatorios, etc.

[56]El análisis basado en la oración es cuando analiza el texto y luego ora para que lo entienda correctamente. p.ej. Cortar las primeras 3 (¿o eran 4?) Líneas, recortar la columna 30-40 (suponiendo que esos espacios no son Tabs), convertir a un entero (hmm. - ¿Alguien utiliza 64 bits? ... bueno espero que sea de 32 bits).

[57]Un "objeto" en este sentido es poco más que un conjunto de datos estructurados, a diferencia de una tabla de base de datos o una hoja de cálculo. Cada objeto representa algún componente de gestión, y sus propiedades representan bits de información sobre ese objeto. Los comandos no tienen que analizar estos objetos para encontrar datos, ya que .NET entiende la estructura del objeto y puede simplemente recuperar los bits de información haciendo referencia a los nombres de propiedades.

[58]http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconreflectionoverview.asp

y métodos que tiene, obtener los valores de propiedades e invocar sus métodos).

El entorno de tiempo de ejecución de Monad proporciona un medio para acceder a los cmdlets y ejecutar secuencias de comandos en máquinas remotas a través de Web Services. [59]

4.3 - Un nuevo enfoque de los modelos de gestión

El enfoque tradicional de los modelos de gestión produce una experiencia de administración inconsistente. Hoy en día hay miles de comandos optimizados localmente. Cada desarrollador de comandos define su propio modelo de gestión con un conjunto de nombres y conceptos. Mientras se produce la copia de comandos populares, no hay incentivo sistémico para hacerlo. Se han hecho esfuerzos para proporcionar directrices que impulsen la optimización global, pero el peso del legado ha dificultado que tales esfuerzos ganen mucha fuerza.

Una situación similar existe con las tecnologías de instrumentación actuales que languidecen debido a la falta de soporte de herramientas. Los esfuerzos de evangelización por instrumentación son difíciles a medida que los grupos [de productos] rechazan la estrategia "construye y vendrá". Los desarrolladores de herramientas se resisten a la vasta superficie de los objetos y responden proporcionando una funcionalidad genérica (como supervisión o navegación) a través de una amplia gama de objetos o proporcionando funciones complejas para un pequeño conjunto de problemas.

Monad adopta un enfoque diferente: minimiza el coste de la automatización y proporciona un beneficio inmediato para el usuario

[59] Una de las primeras referencias directas a lo que se convirtió en PowerShell Remoting, que de hecho es un servicio web basado en WS-MAN (Web Services for Management).

final proporcionando **clases de extensión de automatización** basadas en escenarios y herramientas in-the-box que explotan esas clases. Monad puede soportar casi cualquier esquema de automatización, pero alienta firmemente el uso de esquemas estándar proporcionando un conjunto de clases base para escenarios administrativos específicos. Estas clases base incluyen: Navegación, Diagnóstico, Configuración, Ciclo de Vida y Operaciones [60]. Dichas clases proporcionan una sintaxis común, conmutadores, mensajes de error internacionalizados y soluciones a problemas de escenarios comunes (por ejemplo, una implementación común de una pila de directorios para todos los comandos de navegación). Monad también proporciona un conjunto de controles de interfaz de usuario y herramientas que se suministran con el sistema operativo que controla dichas extensiones para realizar una tarea de gestión específica

4.4 - Un Nuevo Enfoque a las Herramientas GUI de Gestión

El enfoque tradicional de las GUI de administración proporciona un mínimo de apalancamiento para desarrolladores. Las herramientas de GUI de administración de Windows de hoy en día se desarrollan de la misma manera que una aplicación completa. Tienen código de interfaz gráfica de usuario, aplicación de lógica de dominio/restricción y acceso de API a objetos administrados locales y remotos. Los servicios de GUI de gestión se limitan en gran medida a un contenedor de interfaz de usuario que facilita la multiplexación de varias herramientas y un cierto nivel de integración. Este enfoque requiere un esfuerzo significativo y un conjunto de pruebas exhaustivo.

[60] PowerShell nunca tuvo clases bases específicas para estos escenarios, pero este fue el origen de la lista estandarizada de verbos de PowerShell que se utilizaron en los nombres de cmdlet. Este concepto también impulsó la creación de las abstracciones PSProvider y PSDrive, en el que cualquier almacén de datos podría ser expuesto como una "unidad de disco", lo que permite un conjunto estandarizado de comandos para manipular cualquier almacén de datos expuestos.

Dado que gran parte de la lógica del dominio y la imposición de restricciones está incrustada en la GUI, es común que las líneas de comandos expongan un subconjunto de las funciones de una GUI. El enfoque tradicional funciona en contra de la automatización.

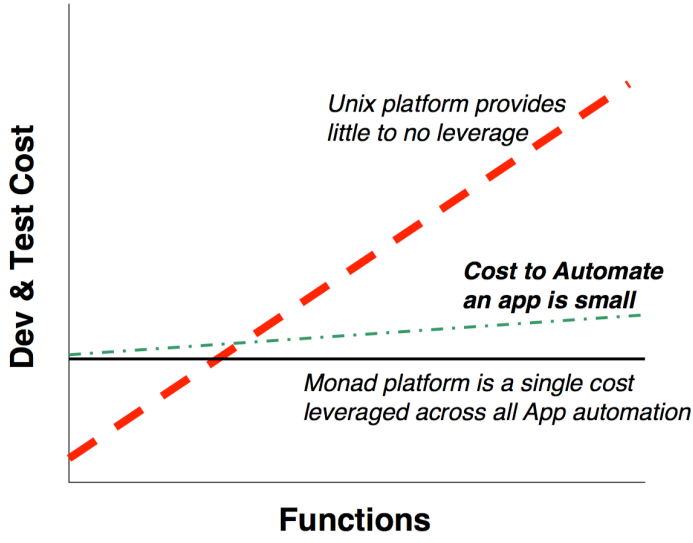
Monad adopta un enfoque diferente que proporciona un rico conjunto de servicios orientados a la gestión para desarrollar herramientas de GUI de gestión. Estos servicios permiten que las GUI de administración se superpongan al motor de secuencias de comandos y Cmdlets. Esto proporciona auditoría, grabación/reproducción de macros y herramientas integradas de GUI/línea de comandos. Esto disminuye el nivel de habilidad requerido para desarrollar una GUI de administración, al simplificar el acceso y el control de los objetos de administración transparentes de manera remota. También permite a los usuarios ver los scripts ejecutados por las interacciones GUI que les ayuda a aprender la capa de automatización y crear sus propios scripts automatizados. La estratificación reduce la matriz de pruebas aprovechando las pruebas realizadas en la línea de comandos y las secuencias de comandos y solo es necesario probar las rutas GUI para invocar esas funciones. La GUI de administración también puede exponer su funcionamiento interno a través de Cmdlets que proporciona a los desarrolladores, probadores y soporte un fácil acceso al estado interno y el control de la GUI para la depuración/diagnóstico/prueba automatizada.

Notas: [^4-1]: ORIGINAL: UNIX tiene getopt ()[61] para el análisis simple de opciones de comandos.

[61] http://www.gnu.org/software/libc/manual/html_node/Using-Getopt.html

Capítulo 5 - El modelo de automatización de Monad (MAM)

Monad define un modelo de automatización altamente apalancado para aplicaciones. El modelo extrae funciones comunes para que puedan implementarse una vez en el entorno de ejecución. Esto proporciona tanto apalancamiento para el desarrollador como coherencia para los administradores. El costo incremental para desarrollar y probar funciones específicas de la aplicación es bastante bajo en comparación con los métodos tradicionales.



Los desarrolladores exponen un modelo de automatización a los

administradores como un conjunto de nombres y verbos fáciles de utilizar. El desarrollador las implementa subclasificando un conjunto de clases de automatización base de .NET, marcándolas con atributos de automatización para producir un conjunto de Cmdlets. El motor MSH expone estos cmdlets como un API y un conjunto de comandos. Los administradores y los desarrolladores de herramientas ahora obtienen una forma general de acceder uniformemente a la automatización de todos los aspectos del sistema operativo.

5.1 - Un ejemplo

Imagine al desarrollador que necesita exponer el registro de sucesos de Windows para la automatización de informes. El desarrollador decide cómo estructurar la automatización en términos de sustantivos y verbos ("Get-EventLog"). Monad proporciona una sólida orientación sobre este tema. El desarrollador escribe un CmdLet (en C #, VB.NET, COBOL, etc) para exponer esta función.

Un CmdLet podría verse así [62]:

```
[CmdLet("Get", "EventLog")]
public class EventLogCmdLet : Cmdlet
{  [Parameter(Position=0)]
   public string LogName = "system"; //Default to the system

   Protected overrride void ProcessRecord()
   {    WriteObject( new EventLog(LogName).Entries);
   }
}
```

A primera vista puede parecer que el Admininistrador no va a obtener mucho uso de este código, pero nada podría estar más lejos de la realidad. El uso de los atributos CmdNoun y CmdVerb registra automáticamente este CmdLet como el comando "**Get-EventLog**" con un solo parámetro "**LogName**". El Admininistrador entonces usa este comando junto con un conjunto de comandos de utilidad base para componer escenarios mucho más complejos

[62] Brevemente, durante el desarrollo, los "cmdlets de script" de PowerShell (ahora, "funciones avanzadas") tenían una sintaxis similar a ésta. En C #, el código fuente del cmdlet todavía se parece mucho a esto.

¿Qué está llenado el log de aplicación? [63]

```
$ Get-EventLog application |Group source |Select -first 5 |Format-Table
counter Property
======  ==========
  1,269 crypt32
  1,234 MsiInstaller
  1,062 Ci
    280 Userenv
    278 SceCli
```

Ejemplo 4

¿Por qué MSI Installer está llenando el log?

```
$ Get-EventLog application |Where {$_.source -eq "MsiInstaller"} `
   |Group Message |Select -first 5 |Format-Table
counter Message
======= ================================================================…
    344 Detection of product '{90600409-6E45-45CA-BFCF-C1E1BEF5B3F7}…
    344 Detection of product '{90600409-6E45-45CA-BFCF-C1E1BEF5B3F7}…
    336 Product: Visual Studio.NET 7.0 Enterprise - English - Inter…
    145 Failed to connect to server. Error: 0x800401F0
      8 Product: Microsoft Office XP Professional with FrontPage --…
```

Ejemplo 5

¿El uso de mi registro de eventos es regular a lo largo de la semana?

```
$ Get-EventLog application |Group {$_.Timewritten.DayOfWeek}
counter DayofWeek
======  ========
  1,333 Tuesday
  1,251 Wednesday
    744 Thursday
    680 Monday
    651 Friday
    556 Sunday
    426 Saturday
```

Ejemplo 6

El administrador puede agregar Cmdlets adicionales a la canalización (pipeline) para filtrar sólo aquellos eventos que se generaron el martes y luego averiguar qué eventos ocurren más allá de ese día (`$ Get-EventLog application |Where {$\_.TimeWritten.DayofWeek -eq "Tuesday"} |Group EventID`). Después de haber encontrado el evento más frecuente de los martes, pueden filtrar fácilmente el registro para ese evento y determinar la distribución de dicho

[63] ORIGINAL: "Get-EventLog Application" es proporcionado por el código de ejemplo anterior y el resto proviene de los comandos de base de Monad. " Group source" cuenta el número de objetos que tienen el mismo valor para una propiedad en particular (es decir, cuántas veces apareció una fuente en particular). "Select -First 5" trunca el conjunto de objetos para que sólo tengan los primeros 5. "Format-Table" formatea los objetos y sus propiedades una tabla.

evento a través de los días de la semana. (`$ Get-EventLog application |Where {$\_.EventID -eq 131080} |Group {$\_.TimeWritten.DayofWeek}`)

Monad requiere una pequeña cantidad de código CmdLet [64] para integrarse en el entorno de ejecución y aprovechar su rico conjunto de funciones y utilidades para proporcionar un potente y distinguido conjunto de funciones administrativas. Si bien este ejemplo se centró en una investigación ad hoc, es obvio cómo esta investigación podría conducir a un conjunto de informes nocturnos automatizados. Este ejemplo es un escenario sencillo. Los cmdlets completos necesitarían proporcionar una gama completa de verbos, hacer que las entradas se validen y realizar el manejo de errores. Sin embargo, los ahorros en desarrollo y prueba son dramáticos.

5.2 – Aprovechando .NET

Los desarrolladores utilizan los atributos .NET para descargar el trabajo al entorno de ejecución [65]. La filosofía general de Monad es implementar las cosas una vez y luego usarlas en todas partes. Un rico conjunto de atributos declarativos dirigen el tiempo de ejecución de Monad para realizar acciones en nombre del desarrollador. Esto transfiere la responsabilidad de escribir y probar este código, así como de interactuar con el usuario durante las condiciones de error y producir y localizar mensajes de error.

Monad define los atributos de automatización en las siguientes áreas:

[64] Tenga en cuenta que incluso en este documento, Snover no era coherente acerca de "CmdLet" versus "Cmdlet". Hoy en día, "Cmdlet" es el estándar. Su idea original era enfatizar que un "Cmdlet" no era un "comando completo" con todo el análisis sintáctico, pero no era un comando tradicional implementado. En su lugar, era parte de un comando, con gran parte de la sobrecarga proporcionada por las clases base del motor de automatización.

[65] Significado, un desarrollador .NET puede indicar al runtime .NET que realice ciertas tareas estandarizadas. Esto se ve mucho en PowerShell: por ejemplo, una función puede declarar un parámetro como obligatorio y el shell aplicará ese atributo en lugar de que el desarrollador de funciones tenga que escribir la lógica para hacerlo.

Parsing Guidance	Indican al analizador cómo asignar la entrada del usuario al objeto de petición CmdLet. P.ej. Cómo asignar los parámetros a las propiedades, o si un calificador es obligatorio.
Data Generation	Dicen al shell que procese la entrada del usuario para generar los datos reales. También habrá procesadores para hostnames, ipaddrs, registrykeyames, ProcessNames, etc.
Data Validation	Expresan reglas de validación en los datos de entrada. P.ej. Cardinalidad de los datos, valores min/max de los datos, etc.
Encoding Directives	Transmiten la entrada del usuario procesada como objetos de datos. P.ej. Un CmdLet puede querer una matriz de StreamWriters en lugar de una matriz de nombres de archivo.
Object Processing	Realizan un conjunto de funciones comunes en tipos de datos comunes. P.ej. convertir una cadena a minúsculas, etc.

Visibility/Applicability	Proporcionan predicados para visibilidad/aplicabilidad. P.ej. Los cmdlets se pueden etiquetar con la máquina y las funciones del usuario. Si una máquina no tiene la función de servidor DHCP, los comandos del servidor DHCP no estarán visibles de forma predeterminada.
Documentation	Proporcionan información de utilidades sobre el elemento. P.ej. Ayuda
Test	Proporcionan sugerencias a las utilidades para facilitar la generación automática de matrices de prueba.

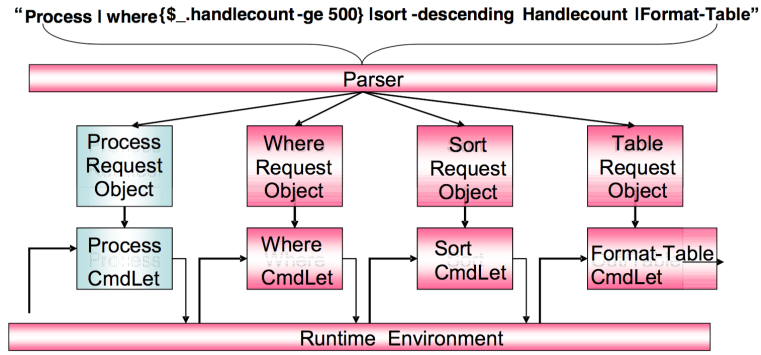
Notas

Capítulo 6 - El Shell Monad (MSH)

Monad proporciona un entorno de tiempo de ejecución para crear APIs, líneas de comandos e interfaces gráficas altamente coherentes, poderosas, detectables y seguras mediante la creación de pipelines de cmdlets. Esta capacidad se suministra como una clase .NET que se puede integrar en una serie de "hosts" que exponen dicha funcionalidad al usuario. El término MSH se refiere tanto al entorno de ejecución como al host que expone el uso como un shell interactivo de línea de comandos.

6.1 - Canalización de objetos (Pipelines) .NET

Monad procesa la entrada del usuario, construye una canalización (Pipeline) de Cmdlets para cada uno de los comandos ingresados, analiza y codifica la entrada del usuario para cada comando en un objeto de petición CmdLet (CRO-CmdLet Request Object). El motor de ejecución de secuencias de comandos "ensambla" la ejecución para luego invocar primer Cmdlet pasando su CRO como un parámetro. Este Cmdlet devuelve un conjunto de objetos .NET que luego se procesan y pasan al siguiente Cmdlet junto con su CRO y así sucesivamente hasta que la Canalización (pipeline) se complete.



Pipelines

Pasar objetos .NET a cmdlets en lugar de flujos de texto permite que las utilidades basadas en Reflection proporcionen una función para cualquier objeto .NET. En el ejemplo anterior, **WHERE** CmdLet filtra un conjunto de objetos basándose en una prueba de las propiedades del objeto. Resuelve objetos de cualquier tipo (por ejemplo, Procesos, Archivos, Discos, etc.) y consultas para su tipo utilizando las API de Reflection de .NET. Mediante el tipo, consulta la existencia de la propiedad especificada por el usuario ("HandleCount"). Luego, utiliza esta información para consultar cada objeto para el valor de dicha propiedad y finalmente realiza la prueba en esa propiedad para filtrar el objeto apropiadamente.

El mismo mecanismo es utilizado por **SORT** CmdLet para ordenar un conjunto de objetos y el **FORMAT-TABLE** CmdLet para mostrar las propiedades de un conjunto de objetos como una tabla. Las utilidades de Monad facilitan el procesamiento de las funciones comunes de los Cmdlets, lo que ahorra costes para el desarrollador y aumenta la potencia/consistencia de los administradores.

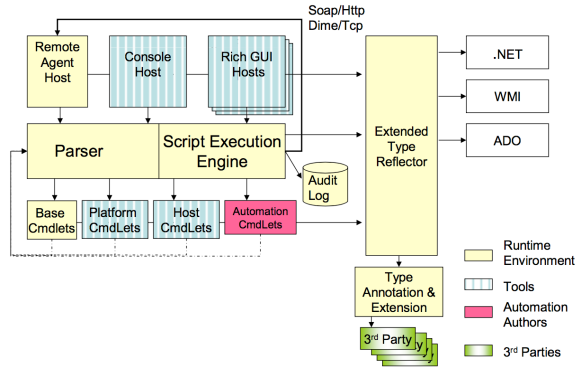
La integración de comandos heredados [66] es trivial porque los flujos de texto son simplemente un tipo de flujo de objetos .NET. Dicho

[66]ORIGINAL: MSH podrá invocar de forma transparente los comandos heredados y los shells heredados podrán invocar sin problemas MSH CmdLets. (MSH proporcionará un mecanismo para exportar CmdLets para el acceso desde los shells heredados) [De hecho, PowerShell nunca implementó una forma fácil de invocar cmdlets para comandos heredados].

esto, una vez traducido a texto, se pierde la capacidad de operar sobre él como un objeto rico basado en Reflection para regresar de vuelta al mundo del análisis de texto.

6.2 - Componentes del entorno de tiempo de ejecución de Monad

El siguiente diagrama ilustra los principales componentes del entorno de tiempo de ejecución de Monad.



Tiempo de ejecución

6.2.1.1 - El analizador

El analizador de Monad es utilizado por todos los Cmdlets y garantiza una sintaxis coherente. Es responsable de analizar la entrada del usuario para el motor de ejecución de secuencias de comandos. Cuando un usuario introduce una línea de comandos, el analizador asigna el comando a un método CmdLet y su Cmdlet Request Object. Los campos y atributos del objeto de petición se utilizan para analizar el resto de la línea de comandos, generar cualquier información adicional, validar la entrada y codificar esos

valores en el objeto de petición.

Al realizar este proceso, el analizador puede agregar metadatos proporcionados por el objeto Request con metadatos proporcionados por proveedores de terceros. Por ejemplo, un objeto de solicitud puede indicar que puede aceptar hasta 16 nombres de nodo y que los nombres deben resolver a una dirección IPv4. Una directiva no puede cambiar esas directivas, pero podría añadir una directiva que indique que los nodos deben responder actualmente a un ping ICMP (por ejemplo, IsAlive).

6.2.1.2 - El motor de ejecución de secuencias de comandos

El motor de ejecución de secuencias de comandos Monad encadena los cmdlets y garantiza una experiencia de ejecución consistente. También es responsable de tomar las canalizaciones (pipelines) codificadas por el analizador y realizar todas las operaciones necesarias para secuenciarlas hasta su finalización. Si las acciones deben ocurrir en una máquina remota o un conjunto de máquinas remotas, se coordina con la MRS[67] y además registra todas las actividades en el registro de auditoría. El motor de ejecución observa el flujo de datos entrante y encuentra las propiedades correctas para enlazar en un CmdLet (un CmdLet puede tener varios parámetros para aprovechar diferentes tipos de datos). La salida de un CmdLet se recoge, se procesa y se pasa a las propiedades apropiadas del siguiente CmdLet. Dado que el entorno de tiempo de ejecución se puede incrustar en varios hosts (por ejemplo, línea de comandos, GUI, etc.), es importante que un CmdLet nunca se comunique directamente con el usuario. El motor de ejecución de secuencias de comandos media esta actividad entre el CmdLet y los distintos hosts.

[67] https://www.penflip.com/powershellorg/monad-manifesto-annotated/blob/master/chapter-8-the-monad-remote-script-mrs.txt

6.2.1.3 - Los Cmdlets

Los cmdlets realizan acciones. Hay cuatro tipos de Cmdlets: 1) Base 2) Host 3) Plataforma y 4) Usuario. Los cmdlets **Base** funcionarán en cualquier entorno .NET, como Sort, Where, Group, etc. Los cmdlets de **Plataform** son aquellos que dependen de una plataforma determinada (XP, Smart Phone o Compact Framework) y no están disponibles en otras plataformas. Los cmdlets de **Host** son aquellos proporcionados por la aplicación que incorpora el entorno de tiempo de ejecución de Monad. Por ejemplo, msh.exe o GUI de administración que exponen Cmdlets específicos para ese host (por ejemplo, Cambiar una fuente, cerrar una ventana, etc.). Los Cmdlets de **User** son los escritos por el usuario. Estos pueden ser escritos en cualquier lenguaje (C #, VB.NET, etc), pero la mayoría se escribirá en MSH (el lenguaje shell).

El identificador único de estos Cmdlets es su tipo .NET (por ejemplo, System.Command.ProcessCmdLet). Aunque este identificador siempre se puede utilizar para invocar el CmdLet, es largo y hostil. Como tal, los autores de CmdLet están obligados a proporcionar nombres amistosos a través de atributos.

Será bastante común y fácil que los Cmdlets de mayor orden se implementen obteniendo un conjunto de datos y luego utilizando el tiempo de ejecución de Monad para invocar un script en esos datos y devolver los resultados de ese script.

6.2.1.4 - El reflector de tipo extendido

El poder de Monad es su capacidad de aprovechar la reflexión (Reflection) de .NET. El problema es que hay objetos que están codificados en formas que despojan el poder de la reflexión. Por ejemplo, cuando se reflejan contra datos como Datables ADO, donde las propiedades están “encapsulados” como columnas. En esto escenario lo que necesitamos son los nombres de las columnas, pero estos se codifican de forma diferente. Un problema similar

existe con WMI, Active Directory y XML. El reflector de tipo extendido está diseñado para abordar estos problemas.

6.2.1.5 - El Sistema de Anotación y Extensión de Tipo

Tratar con los objetos en crudo proporciona a veces demasiada y otras veces muy poca información. Es el trabajo del tipo anotación y sistema de extensión resolver esta paradoja, proporcionando un mecanismo para que terceras partes definan conjuntos de propiedades (por ejemplo, propiedades asociadas con el rendimiento, la configuración, el consumo de recursos o las dependencias) y dar al conjunto un nombre público. Esto permite al usuario dar un nombre en lugar de tener que especificar cada propiedad. P.ej. "Format-Table resources " vs. " Format-Table name ,pid, workingset, handlecount, virtualmemory, privatememory".

Monad proporciona acceso a los objetos y los métodos sobre esos objetos. Sin embargo, los métodos intrínsecos de un objeto representan un número muy pequeño de las cosas interesantes que los usuarios quieren hacer. El mecanismo de extensión de tipo permite que terceras partes registren métodos intermedios en esos objetos. Estos métodos pueden ser accedidos utilizando la misma sintaxis que los nativos, pero este sistema los enviará al método apropiado de terceros, pasando el objeto original como un parámetro.

6.2.1.6 - El Agente Remoto

Los usuarios podrán ejecutar secuencias de comandos en máquinas remotas a través de solicitudes de servicio Web para el host de agente remoto. Este host incorporará el tiempo de ejecución y responderá a las solicitudes recibidas vía SOAP/HTTP o DIME/TCP. Los usuarios serán autenticados y sus actividades autorizadas (ya sea por ID o por ROLE). Las solicitudes y las respuestas se codificarán de manera que permitan la cancelación y el rastreo de las actividades locales a solicitudes específicas en los registros de auditoría remota.

Cuando una secuencia de comandos está completa, sus objetos de retorno se serializan por valor para la transmisión a través de la red.

6.2.1.7 - Seguridad

Monad podría ser uno de los entornos de shell más seguros jamás creados. Todas las acciones se consignan en un registro de auditoría. Las facilidades de identificación de código proporcionadas por .NET reducen significativamente la exposición a una de los problemas de seguridad más comunes en un entorno de shell: Troyanos. Firmas, nombres fuertes y hashes en la política del sistema se utilizarán para identificar qué utilidades son legítimas y están aprobadas así como para evitar que troyanos conocidos se ejecuten.

En resumen, el shell de Monad minimiza las exposiciones de seguridad y facilita la detección y corrección de las brechas de seguridad.

6.2.1.8 - Host MSH

MSH es un ensamblado .NET que se puede incrustar en cualquier host ejecutable para proporcionar ejecución de scripts y acceso a Cmdlets. Los hosts son capaces de determinar qué subconjunto de Cmdlets se ponen a disposición del usuario. El caso más común es que un Host expone todos los Cmdlets de Base (por ejemplo, ordenar, filtar, etc.), todos sus Cmdlets de Host (por ejemplo, Outlook expondría Cmdlets para tratar con buzones y mensajes) y un subconjunto apropiado de los Cmdlets de Plataforma Cmdlets que tratan de procesos, discos, adaptadores de red, etc.).

MSH es también un ejecutable autónomo que aloja el motor de ejecución de secuencias de comandos y proporciona una rica experiencia interactiva al mismo tiempo que una experiencia de tipo vt100 convincente. MSH proporciona capacidades gráficas complejas como Intellisense para completar el comando. Los datos se pueden

imprimir en formatos gráficos para aprovechar las capacidades de interacción y visualización de las PC.

6.3 - Lenguaje de secuencias de comandos de MSH

MSH proporciona un lenguaje de scripting completo utilizando las funciones y la sintaxis del modelo POSIX Shell (control de flujo, manejo de fallas, variables, definición de funciones, alcance, redireccionamiento IO, etc.) como punto de partida, para mejorar la experiencia de programación, aprovechar la nueva funcionalidad o proporcionar una ruta de evolución a C#. El objetivo es que los administradores de UNIX que trabajen con Windows encuentren fácil aprender y migrar sus habilidades a MSH.

Además de escribir funciones tradicionales, los usuarios pueden usar las capacidades de secuencias de comandos de MSH para escribir sus propios Cmdlets y para agregar o reemplazar verbos a los sustantivos existentes de CmdLet.

Notas

Capítulo 7 - Modelos de gestión de Monad (MMM)

Monad ayuda a los desarrolladores de aplicaciones a diseñar la experiencia administrativa proporcionando un conjunto de modelos de gestión. Un MMM es un conjunto rico de clases base de automatización basadas en escenarios y una herramienta o conjunto de herramientas que utilizan esas clases para realizar un escenario de administración particular. Estas clases bases cubren los principales escenarios de administración, incluyendo: Navegación, Diagnóstico, Configuración, Ciclo de vida y Operaciones. Las clases base proporcionan una forma común de realizar estas tareas en varios tipos de recursos. Esto permite al administrador aprender un modelo para gestionar un escenario en particular y luego aplicar ese modelo a una amplia gama de problemas y nuevas situaciones. Los desarrolladores seleccionan el conjunto apropiado de clases base, derivan sus propias clases de éstas e implementan los métodos apropiados para sus tipos de recursos. Las clases base proporcionan lo siguiente:

1. Un conjunto de verbos para el escenario (por ejemplo, Navigation tiene el conjunto de verbos: pwd, cd, dir, pushd, popd, dirs)
2. Un conjunto de objetos de solicitud base que definen calificadores comunes. P.ej. Si el escenario se refiere a una máquina remota, el objeto de petición base definiría un calificador común -MACHINENAME. Esto disuade a la gente de usar los términos: NODE, SERVER, HOST, etc.
3. Un conjunto de excepciones y mensajes de error para ese escenario. P.ej. Habrá una excepción esquematizada estándar

para “Recurso no disponible” para que no terminemos con decenas de variaciones [que existen hoy en día].

4. Soluciones comunes a problemas de escenario comunes. P.ej. Las clases base proporcionarán una solución estándar para el problema de que alguien pida accidentalmente demasiada información [por ejemplo, obtener todos los objetos en LDAP].

Microsoft localizará todas las partes visibles de estos escenarios (verbos, calificadores, mensajes de error, etc.) para que los ISVs puedan reducir significativamente sus costos de desarrollo mediante el aprovechamiento de estas clases base. Además de estos beneficios, Monad proporciona controles de interfaz de usuario para mostrar gráficamente e interactuar con implementaciones de estas clases base. Monad enrá con un plug-in MMC de herramientas que alojan estos controles de interfaz de usuario, pero los ISV o los desarrolladores internos pueden alojar los controles en sus propias UI de administración. Dado que estos controles tendrán acceso a interfaces de datos y control bien definidos, terceras partes también pueden crear controles de reemplazo.

Un ejemplo

La navegación proporciona un ejemplo de un modelo de gestión. Habrá una clase base para todos los Cmdlets que quieran hacer Navigation. Esto definirá los verbos (pwd, cd, pushd, dirs, popd, dir), mensajes de error comunes y proporcionará implementaciones comunes para problemas comunes (pushd, dirs y popd se implementarán una vez). Esa clase base puede ser “subclaseada” para proporcionar una experiencia de administración consistente con una cantidad mínima de código. Una vez que el administrador aprende cómo usar este modelo, podrá utilizarlo en una amplia gama de recursos. Navegar el sistema de archivos será el caso por defecto:

```
[4]$ pwd
F:\xpsh\prototype4\bin
[5]$ dir
Written                Length.kb Name
==================== ========= ================================
5/17/2002 1:02:26 PM          11 audit.txt
5/15/2002 12:56:35 PM         44 AxInterop.SHDocVw.dll
5/17/2002 12:55:28 PM         64 basecmds.dll
5/17/2002 12:55:28 PM        232 basecmds.pdb
[6]$ pushd ..
[7]$ dirs
F:\xpsh\prototype4
F:\xpsh\prototype4\bin
```

Example 1

Los mismos comandos se pueden utilizar para explorar el Registro:

```
[2]$ pwd/reg
HKEY_LOCAL_MACHINE
[3]$ dir/reg
Name                          SubKeyCount ValueCount
========================== =========== =========
HKEY_LOCAL_MACHINE\HARDWARE           4          0
HKEY_LOCAL_MACHINE\SAM                1          0
HKEY_LOCAL_MACHINE\SOFTWARE          32          0
HKEY_LOCAL_MACHINE\SYSTEM             7          0
[4]$ pushd/reg HARDWARE
[5]$ dirs/reg
HKEY_LOCAL_MACHINE\HARDWARE [0x628]
HKEY_LOCAL_MACHINE
```

Example 2

Los mismos comandos se pueden utilizar para explorar el sistema de Ayuda, Active Directory, bases de datos SQL, WMI u otros espacios de nombres.

Capítulo 8 - El Script Remoto de Monad (MRS)

Monad proporciona un mecanismo basado en Web Services para ejecutar scripts en sistemas remotos. Los scripts se pueden ejecutar en un solo o un gran número (muchos miles) de sistemas remotos. Los resultados de los scripts pueden ser procesos a la medida que cada script individual completa o los resultados pueden ser agregados y procesados en masa cuando todos han terminado. Un script se puede ejecutar en modo BestEffort o Reliable. Los scripts de BestEffort se ejecutan desde el proceso existente y si ese proceso termina, no se realiza ningún trabajo para limpiar los scripts remotos y se pierden los resultados. Los scripts de modo Reliable se persisten en una tienda de SQL local y un servicio maneja la ejecución de la secuencia de comandos. El usuario puede iniciar sesión en la máquina y el servicio continúa procesando el script. El usuario puede iniciar sesión y obtener los resultados de ese trabajo en el futuro.

Capítulo 9 - La consola de administración de Monad (MMC)

Monad proporciona un conjunto rico de Cmdlets de servicio de marco de administración para facilitar la creación de consolas de administración. Estos servicios reducen los costos de desarrollo y de pruebas para producir UIs y consolas de administración, a la vez que permiten una experiencia integrada y administrativa. Los servicios se utilizan para producir una consola de administración integrada, pero también pueden ser utilizados por terceros o por la propia TI para implementar su propia consola de gestión. El objetivo es ser capaz de proporcionar el 50-70% de una herramienta genérica de administración GUI de forma gratuita sólo mediante la construcción del tipo correcto de Cmdlets. Monad ofrece los siguientes recursos y servicios:

1. Un entorno de ejecución de secuencias de comandos que proporciona a las GUI un acceso uniforme y coherente a recursos locales y remotos.
2. Una interfaz integrada y un entorno de línea de comandos para que las interacciones GUI se muestren en una consola de línea de comandos. Los usuarios pueden usar esto para aprender la capa de automatización y también pueden ejecutar directamente acciones de línea de comandos. Este mecanismo además proporciona el soporte para la grabación/reproducción de macros.
3. Scripts de aplicaciones específicas. La aplicación puede exponer sus funciones internas (por ejemplo, botones, pantallas, estructuras de datos internas, etc.) a través de Cmdlets para

permitir scripting específico de aplicaciones, depuración y compatibilidad.

4. Controles de la interfaz de usuario base asociados con MMMs específicos. (Por ejemplo, controles de navegación, controles del ciclo de vida, controles de diagnóstico).
5. Conjunto rico de mensajes de error de base que se localizarán por MMC.
6. Un marco de interfaz de usuario declarativa para permitir GUI de administración personalizada basada en metadatos.

Chapter 10 - Value Propositions

- For **application developers** who need to expose their administrative functions as command lines and GUIs, Monad provides a highly productive development framework.
  - Unlike building stand-alone command lines, Monad provides most of the common functions including a parser, a data validator/encoder, error reporting mechanisms, common functions like sorting/filtering/grouping/formatting/outputting and a set of management models which provide common verb sets, error messages and solutions to common problems and tools.
  - Unlike WMI/WMIC, Monad provides a simple programming model. Cmdlets are merely attributed .Net classes.
  - Unlike MMC, Monad provides strong guidance on how to perform management tasks and large benefits (reduced coding/testing) for those that follow that guidance.
- For **application testers** who want to ensure that the administrative command lines and GUIs operate correctly, Monad reduces the amount of code that needs to be tested and increases the productivity of the test process.
  - Unlike building stand-alone command lines, Monad provides a common implementation of most common functions minimizing the amount of application code to develop and test.
  - Unlike traditional management GUIs, Monad layers GUIs on top of Cmdlets so the bulk of the GUI core will already be tested when the command line is tested. Monad will also make it easier to test GUIs by exposing

the inner workings of the GUI through a command line shell and by the ability to drive the GUI controls and code paths through command line scripts.

- For **power users** who want to interact with the system through command line interfaces, Monad provides a highly consistent set of commands and utilities as well as an environment that allows the creation of custom admin tools (i.e. not scenario bound).
  - Unlike cmd.exe, sh, ksh, csh, etc and traditional commands and utilities, Monad provides a common parser for all CmdLet and utilities ensuring syntactic consistency and common input error handling and messaging across all Cmdlets and utilities.
  - Unlike cmd.exe, sh, ksh, csh, etc and traditional command and utilities, Monad provides a strong prescriptive guidance and enforcement of CmdLet naming and error handling and provides a set of scenario automation base classes which make it easy and valuable for developers to follow those guidelines.
  - Unlike cmd.exe, sh, ksh, csh, etc and traditional command and utilities, Monad replaces pipelines passing text with pipelines passing .Net objects which allows utilities to use the .Net reflection APIs to operate directly against the objects without the need to perform error-prone text parsing and object lookup.
- For **Administrators** that want to develop management scripts to automate the management of their systems, Monad provides a highly productive model for learning and effecting that automation.
  - Unlike cmd.exe, the Monad shell is based upon and extends the Bourne Shell syntax and control structures facilitating the skill transfer of Unix Admins.
  - Unlike sh, ksh, csh, etc and traditional command/utilities, Monad uses .Net objects instead of text as an

integration mechanism allowing easier and more precise integration.
  - Unlike sh, ksh, csh, etc and traditional command/utilities, Monad exposes a rich error model leveraging .Net objects to expose precise details of what went wrong, where, when, and what objects where processed/unprocessed.
  - Unlike traditional management GUIs, Monad GUIs allow Admins the ability to see the inner workings of the GUI by exposing their actions via a command line console so that the Admin can learn the automation surface by using the GUI.
- For **GUI users** who want to automate their operations, Monad facilitates learning the automation layer by exposing the shell equivalents of GUI interactions.
  - Unlike traditional management GUIs, Monad GUIs are layered on top of Cmdlets so every function available in the GUI is also available via the command line. Unlike traditional management GUIs, Monad GUIs allow Admins the ability to see the inner workings of the GUI by exposing their actions via a command line console so that the Admin can see the command line equivalent of their GUI interactions.