Design and Implementation of Manageability Services for Common Management Model

C. Eric Wu

IBM T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598
cwu@us.ibm.com

Abstract

Grid services are emerging technologies for the next generation web services. In this paper we design and implement a manageability framework based on Globus toolkit version 3. Two port types, one for enumerating resource ids and the other for enumerating associated resource ids, are proposed for Common Management Model and implemented in the framework. A number of Grid-enabled manageability services are developed for some of the most commonly used Linux resources, including disk partitions, Linux OS, and Linux processes. Various service operations and service data elements are implemented to enable manageability functions for the resources. Visualization panels are also developed to access these manageability services through the Globus service browser. The on-demand feature of Grid services distinguishes these services from enumeration based systems in which object instances are often created but never accessed.

.

1. Introduction

In today's fast moving business, dealing with system management and robustness is frequently an afterthought. This is acceptable when an organization tries to get its footing by delivering a working system. However, effort should occur in parallel to identify an architecture that enables growth. Good software practices and strong understanding of distributed computing and management issues are essential, and there is no substitute for architecture analysis and planning, as it is crucial for building and managing enterprise information systems.

The development of raw computing power coupled with the proliferation of computer devices has grown at exponential rates in recent years. This phenomenal growth along with the availability of the Internet have led to unprecedented levels of complexity, brought on new challenges for system management, and added demands for skilled IT professionals. Managing vast amount of heterogeneous computing resources is never an easy task, especially when the systems at hand are increasingly distributed. While some resources may be inside the network of an organization, others can be spread across the globe and dynamically connected through the Internet. It is evident that increasing processor power, storage capacity and network connectivity must report to some kind of authority if one expects to take advantage of their full potential. As the total cost of ownership (TCO) is increasingly dominated by human costs, it becomes critical to automate system management to reduce the TCO. This in turn requires software-to-software communication.

Grid services are web services with service data elements (SDEs) [1, 2] that conform to a set of conventions expressed as Web Service Description Language (WSDL) [3] interfaces and behaviors, such as notification, on-demand factory, and lifetime management. They are self-describing, in that WSDL is used to describe operations and service data elements. Service clients do not need to have prior knowledge about operation APIs from Grid services. When a service client accesses the end point of a Grid service, the client learns from the WSDL descriptions of the service before invoking service operations. As in web services, WSDL descriptions eliminate the potential problem resulting from changes in operation API, thus making Grid services very suitable for software-to-software communication. A WSDL file is a text-based XML document, which eliminates byte-ordering problems that are typically associated with binary-oriented remote procedure calls and is therefore allowed to go through corporate firewalls through HTTP requests and responses. In addition to SDEs, Grid services typically provide factories for on-demand services, notification mechanism for information exchange, and use registry for service discovery. These features make Grid services a compelling foundation for resource management across the Internet.

The Common Management Model Work Group (CMM-WG) [4] was established to provide a set of port types that are broad and general use for management in the Grid. These port types will be described in WSDL and built on the OGSI specification [5]. This is a milestone in that a document-centric approach is used for management for the first time.

In this paper we present a preliminary framework for Manageability services. Manageability services are Grid services for managing system resources. They are built using Globus toolkit version 3 – the first major implementation of Grid services and the OGSI specification version 1.0. We propose two port types, one for enumerating resource ids and verifying a given id, the other for enumerating associated resource ids. These port types are needed to preserve the on-demand feature of Grid services, and are broad and general enough to be considered as CMM port types. Port type panels are developed for individual port types in Manageability services and factories. This extends the service browser in the Globus toolkit to interact as a service client. The design and implementation of Manageability services will be discussed and illustrated in following sections.

2. Manageability Services for Management

One of the early open standards for network management, the Simple Network Management Protocol (SNMP) [6] from the Internet Engineering Task Force (IETF) was introduced in 1988 for managing TCP/IP networks. The Web-Based Enterprise Management (WBEM) initiative [7], including the Common Information Model (CIM) [8] and promoted by the Distributed Management Task Force (DMTF), is also evolving as a standard since 1996. The Java Management Extensions (JMX) [9] is yet another interesting development since the late '90s, particularly for Java platforms.

Proprietary platform-specific tools for resource management were the main stream before open standards were developed. Tools implementing open standards may or may not be compatible with one another while competing for market shares. Organizations often acquire individual management tools for specific platforms over time, resulting in having multiple tools for managing various platforms. The constant need to upgrade these tools and educate administrative staff, coupled with the proliferation of computing devices, may easily lead to a skyrocketing TCO. This warrants the need for an open, service oriented, highly scalable, standards-based integration model for management. Grid services, derived from web services for heterogeneous environments, provide the necessary infrastructure to integrate resource management functions with different platforms offering different APIs and implementations.

Open standards have the advantage of being potentially supported by multiple vendors over proprietary systems. The use of open standards facilitates the management of widely heterogeneous systems and networks, and allows one to exploit the work of other organizations. Grid services, as the emerging standard for "stateful" web services, are likely to flourish along with the Internet.

Grid services use WSDL to describe operations and service data elements. A WSDL file typically contains a collection of description components that apply within a single target namespace. A description component is a description of some aspect of a web service. A *message* consists of a collection of typed data items. An exchange of messages between the service provider and requestor is described as an *operation*. A collection of operations is called a *port type*. Collections of port types are grouped and called a *service type*. A *service* represents an implementation of a service type and contains a collection of *ports*, where each port is an implementation of a port type, which includes all the concrete details needed to interact with the service.

Manageability services are Grid services for managing system resources. As for Grid services, many manageability services can be implemented as <*factory, service*> pairs. The factory is responsible for resource id enumeration and instance creation, while a service instance is responsible to manage the underlying system resource.

2.1 Managed Resource Factory Port Type

In Globus Toolkit version 3 Gird services can be created from the default factory. Manageability services, on the other hand, must have the underlying resources to back them up. For example, a manageability service for disk partitions manages a specific disk partition, and it is mandatory that the specific disk partition exists when

the service is created. The default factory port type with the *createService()* operation creates a Grid service but does nothing to help a user identify existing resources. Thus, we define a managed resource factory (i.e. MRFactory) port type to enumerate existing resource ids and to verify if a given resource id is valid. The *enumerateIDs()* operation in the managed resource portType merely enumerates valid resource ids in its factory and does not create service instances. To create a service instance, a user must call the *createService()* operation of the factory port type using one of the valid ids. This on-demand feature prevents users from wasting system resources, especially when they are interested only in one of the many resources of the same type, such as processes. A manageability service factory therefore implements both the default factory portType and the managed resource factory port type.

We use Figure 1 to illustrate the design of a disk partition factory – a persistent Grid service, and the MRFactory port type along with its relationship with other port types. Figure 1 shows the port type hierarchy for the disk partition factory, which is responsible to create and remove disk partitions.

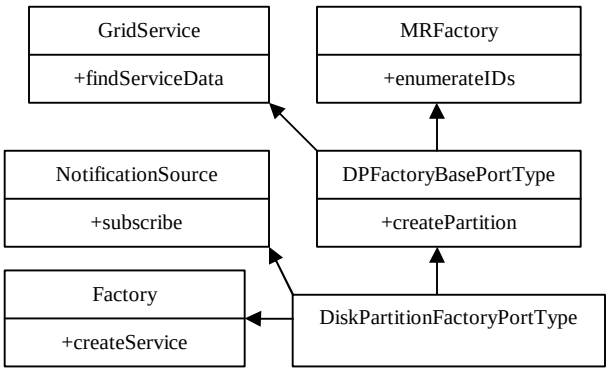

Figure 1. Port type hierarchy for disk partition factory

The disk partition factory service extends the persistent Grid service, implements the default Factory and MRFactory port type operations, and uses the notification source provider for event notification. In order to specify service operations excluding those already defined by the default Factory and Notification Source port types, we use the DPFactoryBase port type that extends MRFactory and Grid Service port types. Thus, the disk partition factory implements operations in the DPFactoryBase port type but not operations in the Notification Source port type. The port type hierarchies for other manageability factories are derived in the same way.

2.2 Association Port Type

A resource may be related to other resources, or in other words "associated with" other resources. For example, a system service such as *sendmail* may be running as a couple of daemon processes. If we want to increase the priority of the daemon processes, it will be helpful to know which processes the *sendmail* system service is currently running on. In other words, it will be helpful if we can get associated resource ids from a given service instance. As a result we define the association port type for manageability services with the *enumerateAssociatedIDs()* operation. Similar to the *enumerateIDs()* operation, it enumerates related resource ids for the given resource type and does not create service instances. A manageability service therefore implements the association portType operation to find associated resource ids for a given resource type. These associated resource ids are actually Grid Service Handles (GSHs) that may or may not exist at the time when the request is made.

Figure 2 illustrates the port type hierarchy of a disk partition service in UML. A disk partition service is used to manage a given disk partition. It defines service operations such as *makeFIleSystem(), mount(),* and *umount().* The disk partition service extends the default Grid service, implements the partition base and association port type operations, and uses the notification source provider for event notifications. Port type hierarchies for other manageability services are derived in the same way. A manageability service port type extends the notification source port type and its own base port type, which in turn extends the Grid service port type and the association port type. It takes advantage of the notification source provider and eases the task of implementing manageability services.

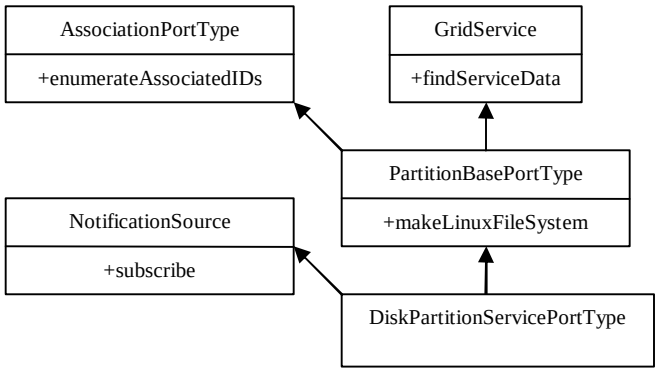

Figure 2. Port type hierarchy for disk partition manageability service

3. Implementing Manageability Services

Manageability services are Grid services for managing system resources. We implement a prototype collection of these services for some most commonly used Linux resources, such as Linux disk partition, Linux OS, and Linux processes.

3.1 Disk Partition Service

Disk partition service is one of the first manageability services developed to show the feasibility of using Grid services for system management. In addition to the inherited portTypes (GridService port type, Factory port type, and MRFactory port type), the disk partition factory implements its base port type, which defines the *listPartition(), createPartition(),* and *removePartition()* operations. Two service data elements are defined in the factory: *Disks* and *PartitionInformation*. The *Disks* service data element is an array of information items expressed in XML, one for each disk to specify information such as the device, number of cylinders, disk size, etc. The *PartitionInformation* service data element is basically the XML expression for the output of the *listPartition()* operation. It is an array of partition information items, one for each disk partition to specify the device, start cylinder, end cylinder, system id (0x82, 0x83, 0x5, etc) and name (Linux, Linux swap, FAT16, HPFS/NTFS).

For each disk partition service we implemented four operations: *mount(), umount(), makeLinuxFileSystem(),* and *pvCreateForLVM(),* as defined in the disk partition portType. The *pvCreateForLVM()* operation initializes the partition for use with Logical Volume Manager (LVM). It checks the system id of the partition and ensures that it is set to 0x8e (Linux LVM). The *makeLinuxFileSystem()* operation takes three input parameters: the name of the file system such as ext2 or ext3, the category of how the file system is going to be used, and an optional label. It would also change the system id of the partition to 0x83 for Linux, if necessary. The category could be “news”, “largefile”, or “largefile4”, to indicate the block size each inode represents (4KB, 1MB, or 4MB). Given a mount point, the *mount()* operation mounts the partition if a file system already exists in the partition. The *umount()* operation un-mounts the mounted file system. A service data element, *DiskPartitionState*, is defined in the disk partition service. It is similar to the service data element *PartitionInformation* and has information such as its size in Kbytes, a flag if it is currently mounted, and information on its locations specified in sectors instead of cylinders. If it is mounted, the *MountInformaiton* service data element specifies the mount point, file system, file system size, used size, available size, used percentage, and label. All sizes are in units of Kbytes in *MountInformation*.
Note that we could have separated file systems from disk partitions to have an additional level of abstraction, or define logical disks and/or native logical disks as specified in CIM. On the other hand, it seems that our prototype is adequate for a minimal and reasonable implementation in demonstrating manageability services.

3.2 Linux Process Service

Clients of existing management systems, such as the CIM Object Manager (CIMOM) from the Storage Network Industry Association (SNIA) [10] and Pegasus from The Open Group [11], typically enumerate all resource object instances of certain resource type before selecting one for management. This is often the case because clients may not know what the valid values are for the key properties of the resource type. For resources like processes whose sheer number could be in the hundreds or even thousands in a large server, the overhead could

be significant. By the time another request is made, many processes have been terminated and new ones created, resulting in new resource object instances. Most object instances are created merely for enumeration, thus wasting time and system resources. The on-demand feature of Grid services helps eliminate this kind of unnecessary overhead in manageability services.

The Linux process factory enumerates resource ids along with their commands. This eases the difficulty of picking the right process when creating its corresponding service instance. The name string in the registry for each service instance also includes all its command line arguments for easy identification. The Linux process service implements two operations: *terminate()* and *setParameter()*, as defined in its port type. The *terminate()* operation kills the Linux process, while the *setParameter()* operation takes a <*parameter*, *value*> pair as input and set the parameter of the process accordingly. Valid parameters for a given process include the nice value of the process, maximal number of child processes, maximal number of open files, and maximal real stack size, each of which is also in the *ProcessState* service data.

Although the rlimit data structure of a given process can be modified through the *setrlimit()* system call from inside the process, we need to do it from the outside of the process in the service. As a result we developed a kernel module to access task structures in the kernel and expose the needed parameters through the /proc file system. The kernel module, *sysman.o*, is installed automatically as needed when the factory is activated. It creates an entry in the /proc file system. A *write()* operation or a simple command "*echo <pid>*" to the entry selects the process with process id <pid>. Subsequent *read()* or *write()* operations (or simple *cat* or *echo* commands) to the /proc entry will then read and/or modify the process' priority or its rlimit parameters. With this simple API we are able to change the priority of a given process through its nice value and modify its rlimit data structure. This simple API has proved to be sufficient and the kernel module has been tested on Linux 2.0 to 2.4 kernels, including RedHat Linux 9.0.

3.3 Linux OS Service

The Linux OS service is a persistent service without a factory. It is more complicated than other services because Linux operating system has many tunable system parameters exposed through the /proc file system. The LinuxOS service includes all these parameters and exposes them through WSDL entries as effectors. An OS parameter portType is used to define the *getParameter()* and *setParameter()* functions, and the Linux OS portType is defined for all other operations in the service.

One problem we face is the fact that the set of parameters varies from one Linux system to another, depending on its OS version, installed kernel modules, and packages. Thus, an *autowsdl* program is developed to find tunable system parameters in the Linux OS and generate corresponding entries in the WSDL file for the OS parameter portType. Parameters are grouped into different categories such as kernel, file system, virtual memory, network core, network IPv4, etc. Each category was then compiled into a Java class as one of the generated stubs. For example, the Java class for kernel parameters in our RedHat 9.0 system with a 2.4.20-8 kernel has 32 entries and that for the virtual memory parameters has 8 entries. Java reflection is used in the portType panel to display two combo boxes, one for selecting the category and the other the individual parameter in that category. As a result the OS parameter portType has only two operations, *getParameter()* and *setParameter()*, yet a user does not have to remember any parameter name to access it through the service browser.

We implemented four operations in the Linux OS portType: *shutdown()* to shutdown the system, *reboot()* to reboot the system, *executeCommand()* to execute a given command line which is passed in as an array of strings, and *getLoadAverages()*. The *executeCommand()* operation provides a mechanism for remote command execution. The *getLoadAverages()* operation gives system load averages in the past 1, 5, and 15 minutes, and may be helpful for load balancing and resource allocation purposes.

A persistent heart beat service is derived from the Linux OS service. It has a single operation *getLoadAverages()* along with SDEs for system loads in the past 1, 5 and 15 minutes, respectively. The heart beat service acquires system loads repeatedly. Its sleep time defaults to 1 minute and may be specified through a deployment descriptor. Clients can either subscribe to its SDEs and get notifications, or invoke the *getLoadAverages()* operation. Thus, it provides both push and pull semantics for communication.

5. Summary

Manageability services are Grid services that provide manageability functions for system resources. They leverage the self-describing, on-demand features of Grid services to control resources across the globe. In this study we built a prototype for a manageability framework. It consists of a number of manageability services for Linux resources, including disk partitions, Linux processes, and Linux OS. The prototype package is available for download at IBM Alphaworks, http://www.alphaworks.ibm.com/tech/gems. The on-demand feature of manageability services eliminates unnecessary overhead resulting from object enumerations in existing systems such as SNIA CIMOM and Pegasus.

We proposed the association port type and managed resource factory port type as port types for the Common Management Model to be inherited by services and service factories respectively. They are associated with their service data elements for enumeration results. Inside each service or service factory, instrumentation is implemented through additional port types and service data elements in order to connect a service instance to its corresponding resource. These instrumentations include a kernel module to modify nice values and per-process parameters, utility to detect tunable system parameters, and numerous command line processes on behalf of the services spawn in the background for implementing port type operations or getting service data elements.

All the services described in this study were built using the most recently released Globus Toolkit version 3. Additional manageability services are under development and will become available over time. Meanwhile we will continue the framework development to keep it compatible with future releases of the Globus toolkit. In addition, manageability services may benefit from merging with Container-Managed Persistent (CMP) entity EJBs for security reasons, and from deployment with JMX MBeans and agents to broaden its applicability. While the wide availability of XML processing tools eases the development of Grid services, tools that create skeletons of manageability services from CIM Managed Object Format (MOF) files or UML are desirable to automate part of the development process and speed up the development cycle.

References

1. "Grid Services for Distributed System Integration," I. Foster, C. Kesselman, J. Nick, and S. Tuecke, pp. 37 – 46, IEEE Computer, June 2002.
2. "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," I. Foster et.al., Technical Report, Globus project, http://www.globus.org/research/papers/ogsa.pdf.
3. "Web Services Description Language (WSDL)," E. Christensen et. al., Technical Report, W3C Consortium, http://www.w3.org/TR/wsdl.
4. Common Resource Model Specification, Draft 1, February 17, 2003, http://www-unix.gridforum.org/mail_archive/ogsa-wg
5. "Open Grid Services Infrastructure (OGSI)," Steve Tuecke et.al, http://www.gridforum.org/ogsi-wg/drafts/draft-ggf-ogsi-gridservice-29_2003-04-05.pdf.
6. "A Simple Network Management Protocol," RFC 1067, http://www.ietf.org/rfc1067.txt?number=1067, IETF.
7. "Web-Based Enterprise Management Initiative," http://www.dmtf.org/standards/standard_wbem.php, DMTF.
8. "Common Information Model: Implementing the Object Model for Enterprise Management," W. Bumpus et. al., Wiley Computer Publishing, ISBN 0-471-35342-6.
9. "Java Management Extensions for Application Management," Heather Kreger, pp. 104 – 129, IBM Systems Journal, Vol. 40, No. 1, 2001.
10. "SNIA CIMOM," http://www.opengroup.org/snia-cimom .
11. "Pegasus CIM Object Broker Manual," The Open Group, http://www.opengroup.org/manual/PDF/PegasusManual.pdf.