RC22821 (W0306-088) June 17, 2003
Computer Science

IBM Research Report

Manageability Services for Linux Resources

Ching-Farn E. Wu, *Hariharan Balakrishnan,
***Biju T. Maniampadavathu, William P. Horn**

IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

*IBM Solutions Research Center
Block 1, Indian Institute of Tech.
Hauz Khas, New Delhi 110016
India

Manageability Services for Linux Resources

C. Eric Wu, Hariharan Balakrishnan, Biju T. Maniampadavathu, William P. Horn

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 develop a manageability framework based on Globus toolkit version 3. It consists of a persistent messaging service for notifying users (i.e. system administrators) of critical changes, and a number of Grid-enabled manageability services for some of the most commonly used Linux resources, including disk partitions, Linux OS, Linux processes, system statistics, and system services. Various service operations and service data elements are implemented to enable manageability functions for the resources. System administrators can then subscribe to one or more service data elements using the messaging service. Visualization panels are also developed to access these manageability services through the Globus service browser. The on-demand feature of Grid services distinguishes manageability services from enumeration based systems in which object instances are often created but never accessed.

1. Introduction

The development of raw computing power in recent years coupled with the proliferation of computer devices has grown at exponential rates. This phenomenal growth along with the availability of the Internet have led to unprecedented levels of complexity, brought on new challenges for system administrators to manage and maintain computer systems, 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 resource and system management [1] to reduce the TCO and thus requires software-to-software communication.

Grid services are web services with service data elements (SDEs) that conform to a set of conventions expressed as Web Service Description Language (WSDL) [2] interfaces and behaviors, such as notification, on-demand factory, and lifetime management. Using a set of open standards and protocols Grid services provide the ability to gain access to a vast array of computing resources over the Internet [3]. These resources could be applications and data, processing power, storage capability, or individual physical or logical subsystems and components.

Grid services are emerging technologies based on web services for the next generation of service oriented architecture [4, 5]. 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.

We present a framework for Linux resources based on the Globus toolkit. Background information for resource management and Grid services is discussed in Section 2, and the design and implementation of manageability services is given in Section 3. We then discuss the messaging service in Section 4, followed by the summary in Section 5.

2. Resource Management and Manageability Services

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) from Sun [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. A successful management approach is likely to be an infrastructure based on open standards and supported by multiple vendors, which is the only effective way to manage systems and components in a heterogeneous environment. 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 *portType*. Collections of portTypes are grouped and called a *serviceType*. A *service* represents an implementation of a serviceType and contains a collection of *ports*, where each port is an implementation of a portType, which includes all the concrete details needed to interact with the service.

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 portType with the *createService()* operation creates a Grid service but does nothing to help a user identify existing resources. Thus, a managed resource factory (i.e. MRFactory) portType is used 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, the user can then call the *createService()* operation of the factory portType 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 Linux processes). A manageability service factory therefore inherits both the default factory portType and the managed resource factory portType.

A resource may be related to other resources, or in other words "associated with" other resources. For example, a Linux 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 portType for manageability services with the *enumerateAssociatedIDs()* operation. Similar to the *enumerateIDs()* operation, it enumerates related resource ids for the given resource type (such as Linux process) and does not create service instances. A manageability service therefore inherits the association portType and implements the operation for finding associated resource ids for a given resource type.

Most 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 corresponding resource. PortType panels are developed for individual portTypes defined in manageability services and factories. This extends the service browser in the Globus toolkit to interact as a service client. A messaging service is also designed and implemented. It can be used to subscribe to any number of service data elements in the manageability services. When a subscribed service data changes, the notification source, i.e. the manageability service, notifies the messaging service which in turn sends out a Lotus Sametime instant message. In the following sections we discuss the manageability services in Section 3, and the messaging service is described in Section 4. Summaries are given in section 5.

3. Manageability Services

Manageability services are Grid services for managing system resources. We use Globus Toolkit version 3 alpha 3 in our development. It provides common mechanisms for critical components such as registry, factory, service data, and notifications.

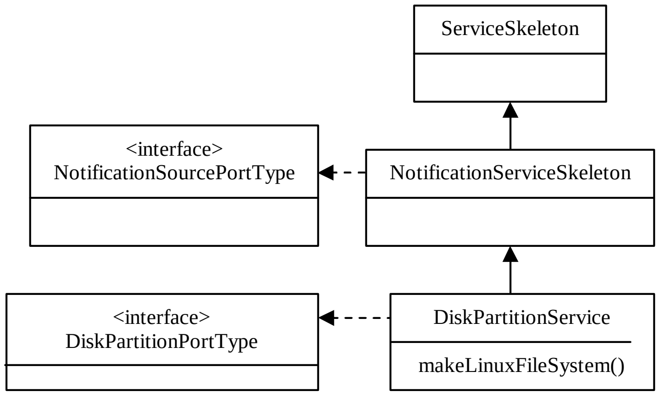


Figure 1. Class hierarchy of manageability services

Figure 1 illustrates a simplified class hierarchy of the implementation for disk partition service in UML. Other manageability services derive from the same notification service skeleton and implement their own port types.

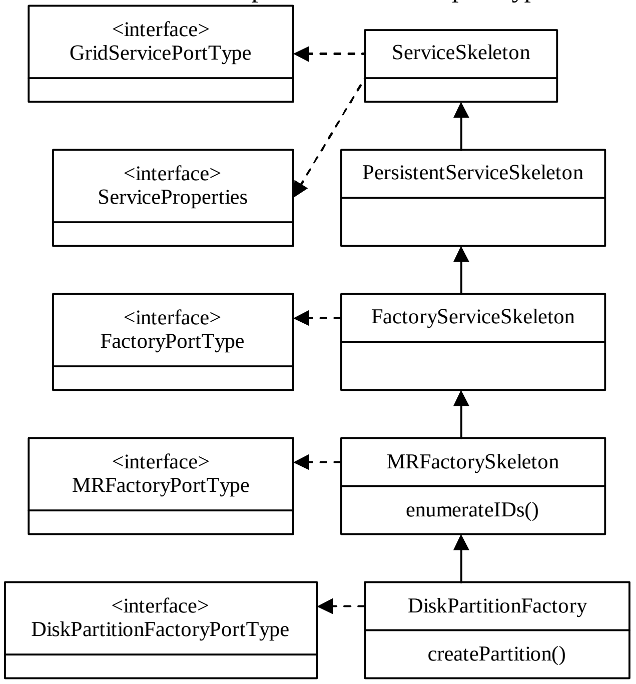


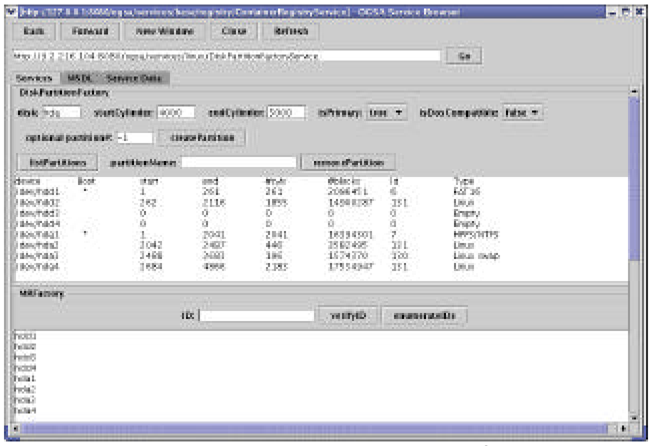
Figure 2. Class hierarchy for disk partition factory service

A simplified class hierarchy of the disk partition factory service is shown in Figure 2. The disk partition factory is derived from the skeleton of the managed resource factory, which in turn extends the default skeleton for factory services. Other manageability factory services are derived in the same way.

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 resource management. In addition to the inherited portTypes (GridService portType, Factory portType, and MRFactory portType), the disk partition factory service implements its own portType, which defines the *listPartition()*, *createPartition()*, and *removePartition()* operations. The *listPartition()* operation returns an array of elements describing the disk partitions currently defined in the system disks. Given the inputs including the selected disk, start cylinder, and end cylinder, *createPartition()* creates a primary or extended partition at the specified location. Unlike the output of the *sfdisk* command in which cylinder number starts from 0, here cylinder number starts from 1 and is compatible with the *fdisk* utility. Thus, an empty entry in a disk partition table will have 0 for its start cylinder and end cylinder.

Two service data elements are also 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 in the disk, 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, etc). Figure 3 shows the service browser with panels for the disk partition factory portType and managed resource factory portType. A list of disk partitions is shown along with the *createPartiion*, *removePartition*, and *listPartition* buttons. The text area in the panel for the MRFactory portType displays all the valid disk partitions.



disk: hdb startCylinder: 4000 endCylinder: 5000 isPrimary: true isDosCompatible: false
optional partition#: -1
partitionName:

```
device     Boot  start  end   #cyls  #blocks    Id   Type
/dev/hda1   *    1      261   261    2096451    6    FAT16
/dev/hda2        262    2116  1855   14900287   131  Linux
/dev/hda3        0      0     0      0          0    Empty
/dev/hda4        0      0     0      0          0    Empty
/dev/hdb1   *    1      2041  2041   16394301   7    HPFS/NTFS
/dev/hdb2        2042   2487  446    3582495    131  Linux
/dev/hdb3        2488   2683  196    1574370    130  Linux swap
/dev/hdb4        2684   4868  2185   17554947   131  Linux
```

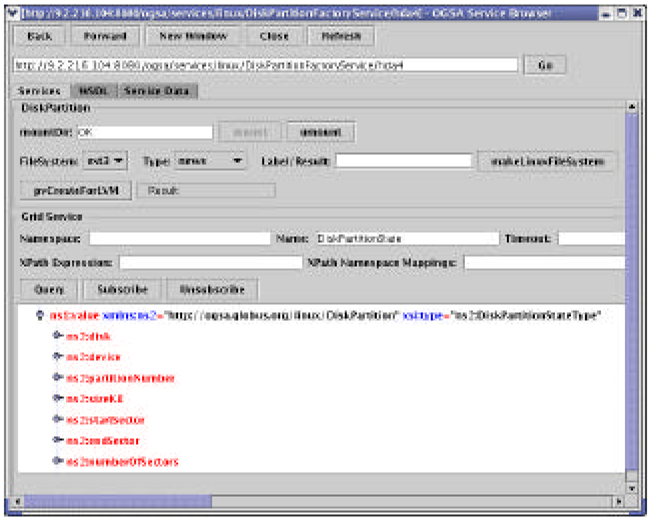
ids:

```
hda1
hda2
hda3
hda4
hdb1
hdb2
hdb3
hdb4
```

Figure 3. Browser snapshot with panels for the disk partition factory portType and MRFactory portType

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 unmounts 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.



```
ns1:value xmlns:ns2="http://ogsa.globus.org/linux/DiskPartition" xsi:type="ns2:DiskPartitionStateType"
    ns2:disk
    ns2:device
    ns2:partitionNumber
    ns2:sizeKB
    ns2:startSector
    ns2:endSector
    ns2:numberOfSectors
```

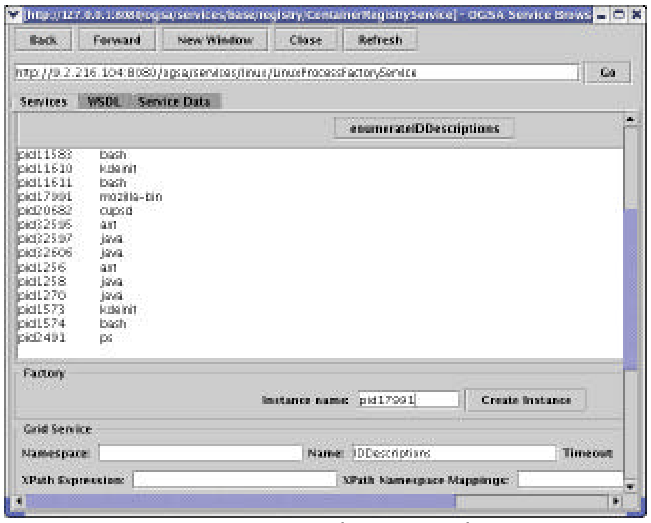
Figure 4. Browser snapshot for disk partition services

Figure 4 shows a snapshot of the service browser for disk partition services. The partition is mounted and its *DiskPartitionState* is also shown in the portType panel for Grid 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, and hence new resource object instances are generated. Most object instances are created merely for enumeration, wasting time and system memory. The on-demand feature of Grid services helps eliminate this kind of unnecessary overhead in manageability services.

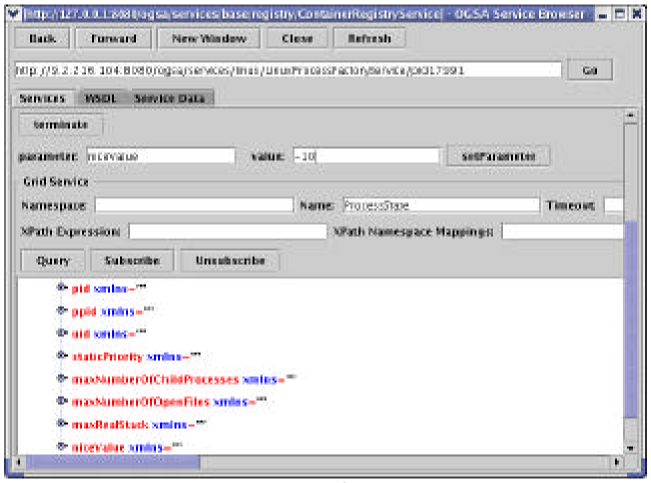
The Linux process factory enumerates resource ids along with their commands. This eases the difficulty of picking the right process when creating its corresponding service instance. The name string in the registry for each service instance also includes all its command line arguments for easy identification. Since processes are created and terminated frequently, our implementation ensures that every request for the enumerated resource ids (i.e. the *EnumeratedIDs* service data) is up to date. This is done through the modification of the server-side stubs. Figure 5 shows a browser snapshot for the process factory service.


[http://127.0.0.1:8080/ogsa/services/base/registry/ContainerRegistryService] - OGSA Service Brows
Back
Forward
New Window
Close
Refresh
http://9.2.216.104:8080/ogsa/services/linux/LinuxProcessFactoryService
Go
Services
WSDL
Service Data
enumeratedIDDescriptions
Factory
Instance name:
pid17991
Create Instance
Grid Service
Namespace:
Name:
IDDescriptions
Timeout
XPath Expression:
XPath Namespace Mappings:

```
pid11583    bash
pid11610    kdeinit
pid11611    bash
pid17991    mozilla-bin
pid20682    cupsd
pid32596    axl
pid32597    java
pid32606    java
pid1256     axl
pid1258     java
pid1270     java
pid1573     kdeinit
pid1574     bash
pid2491     ps
```

Figure 5. Browser snapshot for process factory service

The Linux process service implements two operations: *terminate()* and *setParameter()*, as defined in its portType. 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.



parameter: niceValue value: -10
Namespace: Name: ProcessState Timeout:
XPath Expression: XPath Namespace Mappings:

```
pid xmlns=""
ppid xmlns=""
uid xmlns=""
staticPriority xmlns=""
maxNumberOfChildProcesses xmlns=""
maxNumberOfOpenFiles xmlns=""
maxRealStack xmlns=""
niceValue xmlns=""
```

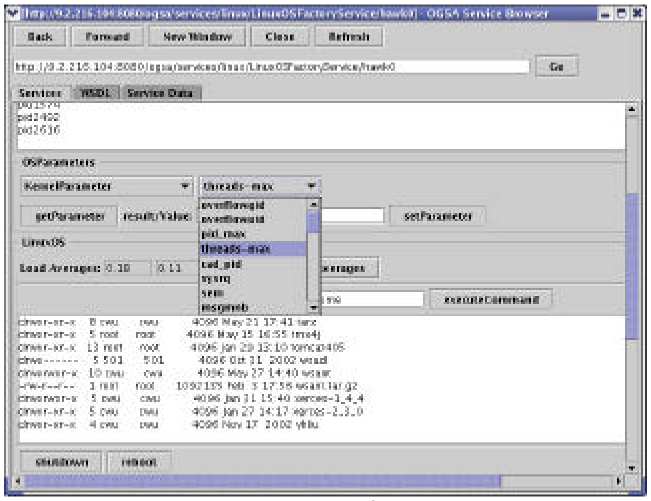
Figure 6. Browser snapshot for Linux process services

Figure 6 shows a browser snapshot for the mozilla process with process id 17991. 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*, 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 could be a persistent service without a factory. To be consistent with other services we choose to use a simple factory that inherits the MRFactory portType but does not define its own portType. Thus the schema path used in the deployment descriptor for Linux OS factory is the service WSDL for managed resource factory services, which is generated from the MRFactory portType. The Linux OS service, on the other hand, is more complicated than other services. This is because Linux operating system has many tunable system parameters exposed through the /proc file system and we decided to include these parameters in the Linux OS service. An OS parameter portType is used to define the get and set functions for OS parameters, 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.



```
pid2400
pid2616
```

KernelParameter threads-max
result/Value:
Load Averages: 0.10 0.11

```
overflowgid
overflowuid
pid_max
threads-max
cad_pid
sysrq
sem
msgmnb
```

```
drwxr-xr-x   8 cwu   cwu      4096 May 21 17:41 [illegible]
drwxr-xr-x   5 root  root     4096 May 15 16:55 [illegible]
drwxr-xr-x  13 root  root     4096 Jan 29 13:10 tomcat405
drwx------   5 501   501      4096 Oct 11  2002 [illegible]
drwxrwxr-x  10 cwu   cwu      4096 May 27 14:40 [illegible]
-rw-r--r--   1 root  root  1092135 Feb  3 17:58 [illegible].tar.gz
drwxrwxr-x   5 cwu   cwu      4096 Jan 11 15:40 [illegible]-1_4_4
drwxr-xr-x   5 cwu   cwu      4096 Jan 27 14:17 [illegible]-2.3.0
drwxr-xr-x   4 cwu   cwu      4096 Nov 17  2002 [illegible]
```

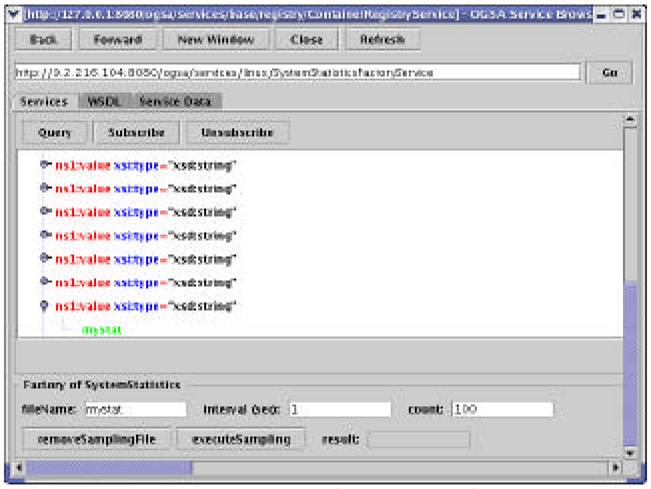
Figure 7. Browser snapshot for Linux OS service

Figure 7 shows a snapshot for the Linux OS service. The text area at the top shows a list of associated processes for the Linux OS. Two combo boxes in the OS parameter portType panel allow a user to get or set any tunable OS parameter. The *getLoadAverages* button, partially covered by the pull-down menu for selecting a kernel parameter, and its returned values are displayed near the center of the snapshot. The bottom half shows the result of executing the "ls –l /home" command, along with the *shutdown* and *reboot* buttons.

3.4 System Statistics Service

While Linux OS, processes, and disk partitions are clearly software or hardware resources, statistics on CPU utilization, paging activity, I/O transfer rate, etc. are critical resources for monitoring system well-being. For this we designed and implemented the system statistics service.

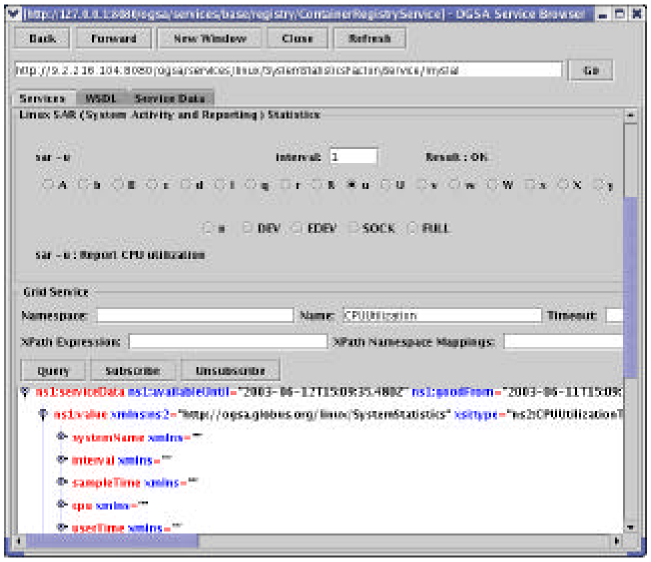
The statistics factory portType has two operations: *executeSampling()* and *removeSamplingFile()*. Given a sampling interval in seconds, the count, and output file, the *executeSampling()* operation starts statistics sampling and store the result in the output file. This is an asynchronous operation and the sampling command is executed in the background. The *removeSamplnigFile()* operation removes the specified file. The only service data defined in the factory is the *EnumeratedIDs*, which is inherited from the managed resource portType and lists all resource ids (i.e. file names) in the factory. Figure 8 shows a snapshot for the statistics factory service. The text area shows the service data *enumeratedIDs*, and the operation buttons are shown at the bottom of the window along with text fields for their inputs and results.



```
[http://127.0.0.1:8080/ogsa/services/base/registry/ContainerRegistryService] - OGSA Service Brows
http://9.2.216.104:8080/ogsa/services/linux/SystemStatisticsFactoryService
ns1:value xsi:type="xsd:string"
ns1:value xsi:type="xsd:string"
ns1:value xsi:type="xsd:string"
ns1:value xsi:type="xsd:string"
ns1:value xsi:type="xsd:string"
ns1:value xsi:type="xsd:string"
ns1:value xsi:type="xsd:string"
    mystat
Factory of SystemStatistics
fileName: mystat    interval (sec): 1    count: 100
result:
```

Figure 8. Browser snapshot for statistics factory service

We use the Linux sar (system activity and reporting) utility to implement the statistics service. Because of the variety of sampling data, the system statistics service has many portType operations, each of which updates its corresponding service data. These operations are required since the background sampling process may be still gathering statistics for the selected sampling file. Service data elements include statistics histories for CPU utilization, process creation, I/O transfer rate, paging, interrupt, network packets received and transmitted, sockets, queue lengths, system loads, memory, memory page and swap space, inode, context switching and swapping, among others.



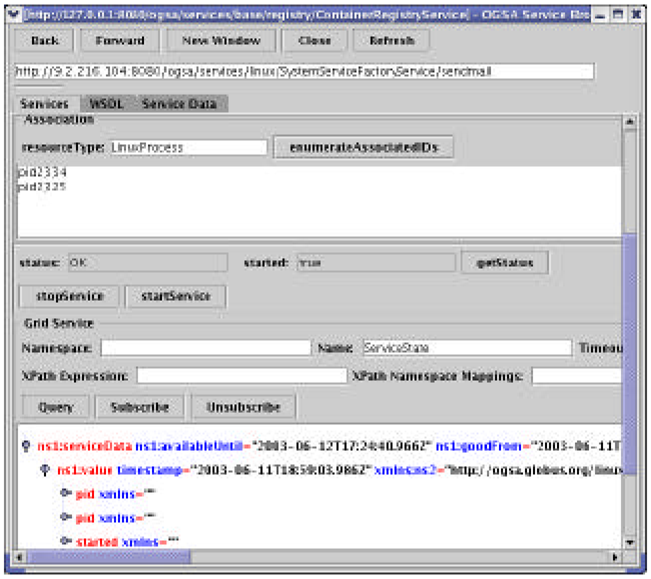
```
[http://127.0.0.1:8080/ogsa/services/base/registry/ContainerRegistryService] - OGSA Service Browser
http://9.2.216.104:8080/ogsa/services/linux/SystemStatisticsFactoryService/mystat
Linux SAR (System Activity and Reporting) Statistics
sar -u                interval: 1        Result : OK
sar -u : Report CPU utilization
Grid Service
Namespace:            Name: CPUUtilization            Timeout:
XPath Expression:            XPath Namespace Mappings:
ns1:serviceData ns1:availableUntil="2003-06-12T15:09:35.480Z" ns1:goodFrom="2003-06-11T15:09:
  ns1:value xmlns:ns2="http://ogsa.globus.org/linux/SystemStatistics" xsi:type="ns2:CPUUtilizationT
    systemName xmlns=""
    interval xmlns=""
    sampleTime xmlns=""
    cpu xmlns=""
    userTime xmlns=""
```

Figure 9. Snapshot for statistics services

Figure 9 is a snapshot for the statistics service. Each radio button corresponds to an update operation in the statistics portType, and labels for descriptions and command line arguments are shown on selection of operation. The text area at the bottom shows the service data *CPUUtilization*, which includes *userTime*, *systemTime*, and *idleTime* percentages at the sample time. Selecting any radio button in the panel updates its corresponding service data.

3.5 Manageability Service for System Services

There are many system services in a Linux system, including *sendmail* for mail daemon, *lpd* for line printer daemon, *syslog* for system logging, *vsftpd* for secured ftp daemon, *crond* for cron daemon, etc. The system service factory inherits the managed resource portType but does not have its own.

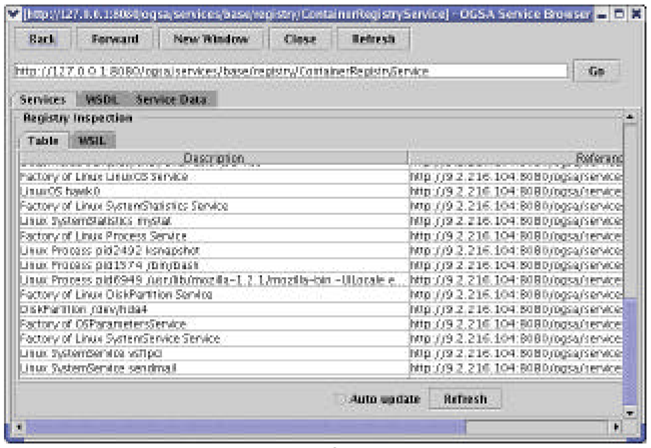


```
[http://127.0.0.1:8080/ogsa/services/base/registry/ContainerRegistryService] - OGSA Service Bro
http://9.2.216.104:8080/ogsa/services/linux/SystemServiceFactoryService/sendmail
Association
resourceType: LinuxProcess
pid2334
pid2335
status: OK            started: true
Grid Service
Namespace:            Name: ServiceState            Timeou
XPath Expression:            XPath Namespace Mappings:
ns1:serviceData ns1:availableUntil="2003-06-12T17:24:40.966Z" ns1:goodFrom="2003-06-11T
  ns1:value timestamp="2003-06-11T18:59:03.986Z" xmlns:ns2="http://ogsa.globus.org/linux
    pid xmlns=""
    pid xmlns=""
    started xmlns=""
```

Figure 10. Snapshot for sendmail system service

The system service portType has three operations: *startService()* for starting the service, *stopService()* to stop the service, and *getStatus()* to get the current status of the service. A service data *ServiceState* is defined in the system service portType and consists of an array of process ids for the service, the service status such as OK or stopped, and if the service is started (boolean). Figure 10 shows a snapshot for the system service representing sendmail. The text area for the association portType shows a list of resource ids for Linux processes associated with the sendmail system service. The *startService* and *stopService* buttons near the center of the snapshot are used to start and stop the service.

Figure 11 shows a snapshot of the Grid service registry in which manageability services for Linux processes, OS, disk partition, system service, and system statistics, along with their factory services. Additional manageability services are under development and will become available over time. Note that individual services for processes are listed with complete command line arguments so that they can be easily identified.



http://127.0.0.1:8080/ogsa/services/base/registry/ContainerRegistryService

Description	Referenc
Factory of Linux LinuxOS Service	http://9.2.216.104:8080/ogsa/service
LinuxOS hawk0	http://9.2.216.104:8080/ogsa/service
Factory of Linux SystemStatistics Service	http://9.2.216.104:8080/ogsa/service
Linux SystemStatistics mystat	http://9.2.216.104:8080/ogsa/service
Factory of Linux Process Service	http://9.2.216.104:8080/ogsa/service
Linux Process pid2492 ksnapshot	http://9.2.216.104:8080/ogsa/service
Linux Process pid1574 /bin/bash	http://9.2.216.104:8080/ogsa/service
Linux Process pid6949 /usr/lib/mozilla-1.2.1/mozilla-bin -UILocale e...	http://9.2.216.104:8080/ogsa/service
Factory of Linux DiskPartition Service	http://9.2.216.104:8080/ogsa/service
DiskPartition /dev/hda4	http://9.2.216.104:8080/ogsa/service
Factory of OSParametersService	http://9.2.216.104:8080/ogsa/service
Factory of Linux SystemService Service	http://9.2.216.104:8080/ogsa/service
Linux SystemService vsftpd	http://9.2.216.104:8080/ogsa/service
Linux SystemService sendmail	http://9.2.216.104:8080/ogsa/service

Figure 11. Snapshot of Grid service registry

4. Messaging Service for Instant Notification

With the proliferation of instant messengers such as Lotus Sametime Connect and those from MSN and AOL, asynchronous instant message delivery has become a reality in today's business world. Instant messengers are so popular that they have become indispensable tools in the IT infrastructure of an enterprise. Leveraging such an infrastructure from a resource management perspective makes it worth investigating.

Notification is the ability to deliver messages from a source to all interested parties. In the OGSI Grid world the sender is called a *notification source* and the receiver is called a *notification sink*. We restrict the scope to notification alone in this section, although we understand that the instant messenger infrastructure can be used for full blown interaction with reference to resource management.

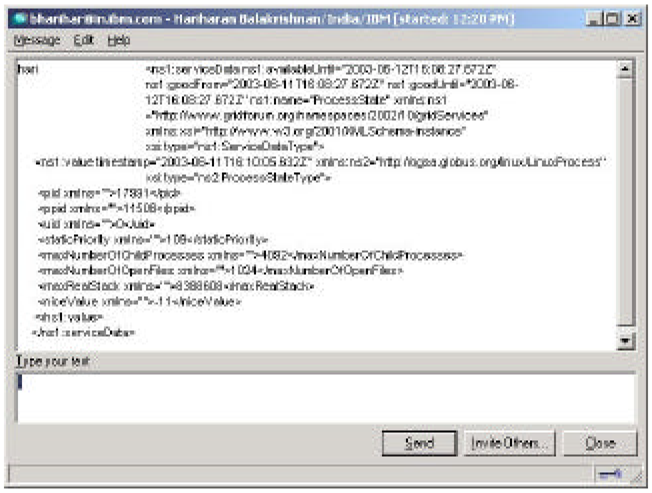
In general notifications are delivered in an asynchronous manner to subscribers through some kind of messaging intermediary software such as Java Message Service (JMS). They could be used to notify users of critical changes in the system, or could be handled directly by some software application to act upon and to do some processing based on the message, as in the spirit of software-to-software communication.

We extend asynchronous message delivery to the world of instant messaging, thereby delivering service data elements to users of Lotus Sametime Connect. Changes in the system may be delivered to a user or a group of administrators so that they can take some actions based on the information. This enhances the interaction between manageability services and their users.

A persistent SameTime messaging service is developed in our manageability framework. This Grid service implements the notification sink portType with the *deliverNotification()* operation for receiving notifications. In addition, a Sametime messaging portType is used to provide a *registerUsers()* operation. The operation takes as parameters a hostName - host address of the Lotus SameTime Server, a user id and password with which the system logs on, and a list of user ids to whom instant messages will be sent. We plan to extend the portType in the future so that each user can specify the type of information or service data he/she is interested in. This could be done through a subscription expression while subscribing to the instant message notification.

To subscribe service data elements in a given service, we invoke the *subscribe()* operation of the notification source portType in the service. The *subscribe()* operation takes parameters such as the Grid Service Handle (GSH) of the Sametime messaging service, the name of the service data element, and the expiration time until which the target Grid service is supposed to send change notification.

The Sametime messaging service, when activated, logs on to the Sametime messaging server using the given user id and password. Whenever the service data element changes in the target grid service to which we have subscribed to, the messaging service will receive an asynchronous notification through the Globus toolkit notification implementation. The Sametime messaging service then invokes the *deliverNotification()* operation of the notification sink portType every time there is a notification. The *deliverNotification()* operation inspects the current list of Sametime users who have subscribed to and delivers an instant message to each of them through Lotus Sametime Connect.


bharihar@in.ibm.com - Hariharan Balakrishnan/India/IBM [started: 12:20 PM]
Message Edit Help
Type your text
Send
Invite Others...
Close

```
hari          <ns1:serviceData ns1:availableUntil="2003-06-12T16:08:27.672Z"
              ns1:goodFrom="2003-06-11T16:08:27.672Z" ns1:goodUntil="2003-06-
              12T16:08:27.672Z" ns1:name="ProcessState" xmlns:ns1
              ="http://www.gridforum.org/namespaces/2002/10/gridServices"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:type="ns1:ServiceDataType">
  <ns1:value timestamp="2003-06-11T16:10:05.832Z" xmlns:ns2="http://ogsa.globus.org/linux/LinuxProcess"
              xsi:type="ns2:ProcessStateType">
   <pid xmlns="">17991</pid>
   <ppid xmlns="">11508</ppid>
   <uid xmlns="">0</uid>
   <staticPriority xmlns="">109</staticPriority>
   <maxNumberOfChildProcesses xmlns="">4082</maxNumberOfChildProcesses>
   <maxNumberOfOpenFiles xmlns="">1024</maxNumberOfOpenFiles>
   <maxRealStack xmlns="">8388608</maxRealStack>
   <niceValue xmlns="">-11</niceValue>
  </ns1:value>
 </ns1:serviceData>
```

Figure 12. An instant message from Lotus Sametime Connect showing a service data element in XML

Figure 12 shows an instant message in which the nickname of the login user is followed by the service data *ProcessState* of the process service with id "pid17991". The service data element is sent in the form of XML, which includes the process id *pid*, parent process id *ppid*, user id *uid*, *niceValue*, etc.

Other service data elements, such as the *CPUUtilization* in statistics services and *DiskPartitionInformation* in disk partition services, can all be subscribed by the messaging service and notifications through Lotus Sametime Connect will be sent to users in the registration list.

The messaging service could be extended using other possible communication channels like the Simple Mail Transfer Protocol (SMTP) thereby an email can be sent to a list of users, or SMS messages to cellular phones etc. Ability to persist and send the message, such as durable subscription, or notifying an alternate user id or notification through email are some of the possibilities which can be incorporated into future versions of the current system.

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 messaging service to notify administrators of critical changes in the system, and manageability services for a number of Linux resources, including disk partitions, Linux processes, Linux OS, system statistics, and system services. The on-demand feature of manageability services eliminates unnecessary overhead resulting from object enumerations in existing systems such as SNIA CIMOM and Pegasus.

Special portTypes such as the association portType and managed resource factory portType are introduced in the framework and inherited by services and service factories respectively. These two portTypes are associated with their service data elements for enumeration results. Inside each service or service factory, instrumentation is implemented through additional portTypes and service data elements in order to connect the service instance to the 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 portType operations or getting service data elements.

Globus toolkit 3.0 beta version was just released. 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. "Autonomic Computing: IBM's Perspective on the State of Information Technology," Paul Horn, http://researchweb.watson.ibm.com/autonomic/manifesto .
2. "Web Services Description Language (WSDL)," E. Christensen et. al., http://www.w3.org/TR/wsdl.
3. "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.
4. "Grid Services for Distributed System Integration," I. Foster, C. Kesselman, J. Nick, and S. Tuecke, pp. 37 – 46, IEEE Computer, June 2002.
5. "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.
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 (JMX)," Sun Micro., http://java.sun.com/products/JavaManagement.
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.