RC22532 (W0207-125) July 29, 2002
Computer Science

IBM Research Report

Building Sensors and Actuators for Adaptive Resource Management in Linux Systems

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

Building Sensors and Actuators for Adaptive Resource Management in Linux Systems

C. Eric Wu
IBM T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Abstract

The foundation of adaptive resource management includes two basic elements: sensors and actuators. Sensors are monitors that collect management data, and actuators are control knobs that are available to change resource configuration, behavior, or allocation. Along with adaptive algorithms, the foundation forms one or more control loops that adapt to environment changes. In this paper we focus on building sensors and actuators in enterprise Linux systems for adaptive resource management. We start by reviewing the current status of industry open standards, including DMTF's Common Information Model (CIM) for system management and Application Response Measurement (ARM), and build sensors for monitoring applications such as the Apache web server. The resulting ARM infrastructure includes a standalone ARM library for application instrumentation, an ARM Agent for collecting and distributing application statistics, a CIMOM ARM Provider, and a web server plug-in that captures URL-based statistics for web servers. For actuators we expose tunable system parameters by building a management tool to provide a consistent and user-friendly GUI, and create new actuators for per-process parameters and entries in configuration files.

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 in recent years coupled with the proliferation of computer devices has grown at exponential rates. This phenomenal growth along with the advent of the internet have led to unprecedented levels of complexity and brought on new challenges for how people manage and maintain computer systems. Demand is already outpacing supply when it comes to managing information systems. This increasing complexity is reaching a level beyond human ability to manage, and it adds demands for skilled I/T professionals at the same time. As the total cost of ownership

(TCO) is increasingly dominated by human costs, system management is a pivotal point in enterprise operations. It becomes essential and inevitable to automate resource and system management to reduce the TCO.

This is the background for "Autonomic Computing" [1], currently advocated by IBM and other leading computer companies and research institutes. It is our belief that increasing processor might, storage capacity and network connectivity must report to some kind of systemic authority if we expect to take advantage of its potential.

In the area of system management, automatic and adaptive resource management is the key for the success of autonomic computing. An adaptive system typically uses some adaptive algorithm to control or reallocate its managed resources through monitoring. Sensors are used to monitor current status and collect management information for resources, and actuators are the control knobs that are available to control or reallocate the managed resources. The control loop, depicted in Figure 1, illustrates sensors and actuators as the foundation of an adaptive management system.

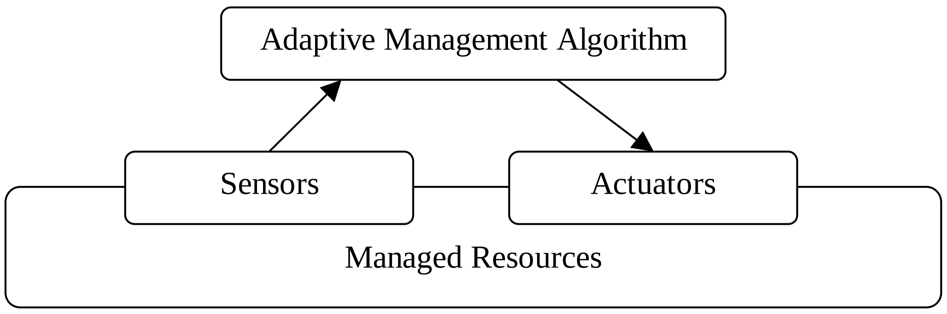


Figure 1. The control loop in adaptive resource management systems

In this paper, we describe our approaches to build sensors and actuators for adaptive resource management in enterprise Linux systems. For sensors we focus on application monitoring and develop our CIM (Common Information Model) based ARM (Application Response Measurement) infrastructure. A CIM-based ARM provider is developed along with an Apache server plug-in as an example to monitor web server performance. For actuators we develop a management utility and its kernel module, to provide a user-friendly tool for control knobs. The management utility is capable of browsing and modifying various aspects of a Linux system, including kernel parameters, per-process parameters, and user accounts. Although it is at least as important for building a goal-oriented adaptive algorithm, we consider it part of our future project and therefore is out of the scope of this paper.

2. Open Standards

The strength of open standards is the potential support from multiple vendors. The railroad, automobile, and telecom industries really didn't take off until the various players agreed on underlying standards, such as what gauge railroad track to use. Was the relatively young IT industry any different? The use of open standards facilitates the management of widely heterogeneous systems and networks, and allows a company to exploit the work of other companies and hence better concentrate on utilizing its resources on exploiting that work. Thus, a good management solution is likely an infrastructure based on standards and supported by multiple vendors – the only effective way to manage systems and components in a heterogeneous environment. Open standards may be the key to success in system management for heterogeneous enterprise environments. Once open standards are developed, they become commodities and everyone can probably get the reference implementation for free. The companies using these open standards differentiate themselves by building outstanding resources.

2.1 Standards for Management

The Common Information Model (CIM) [2] promoted by DMTF is definitely evolving as one such standard. The CIM initiative is an approach in the industry for enabling the management of real world managed objects that applies the basic structuring and conceptualization techniques of the object-oriented paradigm. The CIM standard is composed of four pieces: a language specification for defining real world managed objects, a management schema (core and extension schemas) that provides a common conceptual framework, a set of protocol flows expressed in XML that encapsulate CIM syntax and schema to provide access to real world managed objects, and a compliance document to aid in interoperability between vendor implementations.

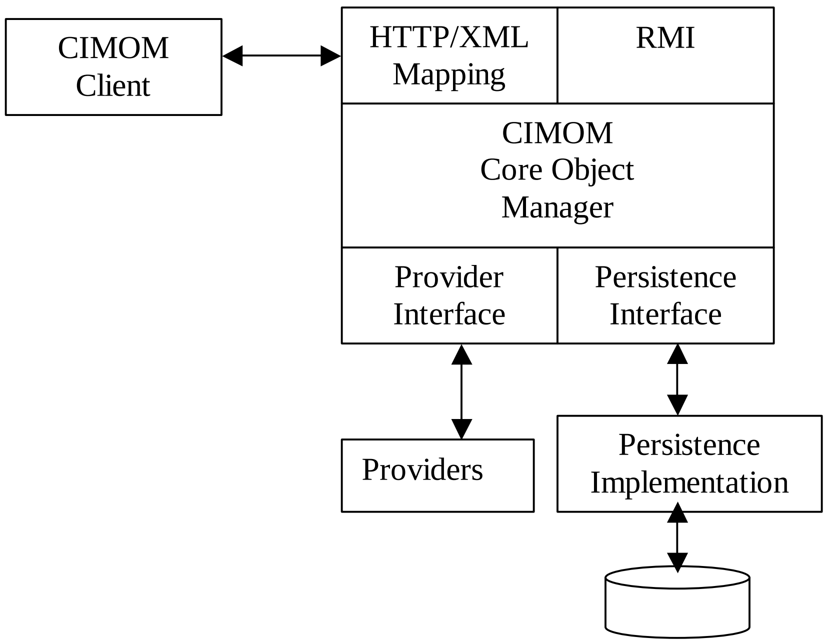


Figure 2. CIMOM and CIMOM client

The industry uses the term CIMOM (CIM Object Manager) to refer to implementations that understand the CIM specification and use some portion of the schemas. These implementations tend to provide their own interface for manipulating the model so the implementations are not a standard. Figure 2 illustrates major components in CIMOM, including the Provider and Persistence interface.

Currently the major players in CIM are Microsoft with its Windows Management Instrumentation (WMI) implementation, Sun Microsystems with its Java CIMOM implementation integrated with its operating system and WBEM implementation as part of a Java standard, Caldera with its C++ CIMOM, and The Open Group with its open source C++ Pegasus implementation [3] and another Java-based open source CIMOM that was originally developed by SNIA [4]. We use the SNIA CIMOM in this paper since it is the first open source CIMOM. However, the content of this paper should be general enough and apply equally well for other CIMOM implementations.

CIM/CIMOM provides a good base in the managed objects already modeled, such as for inventory applications. It has the richness that is needed to define managed resources, such as the ability to express relationships between resources. Either adding new classes or sub-classing from existing classes can extend the CIM schema. The infrastructure without managed resources won't do us any good however, until resources are instrumented through the Provider interface. As the first step towards adaptive resource management, we illustrate our CIMOM ARM provider infrastructure that collects application performance data in the next section.

2.2 Standards for Application Monitoring

The real challenge in system resource management is adaptive management that requires end-to-end performance monitoring. For this, information from within the application program is often needed. The Open Group has defined a couple standards for application management: Application Response Measurement (ARM) [5, 6] and Application Instrumentation and Control (AIC) [7].

The ARM standard describes a common method for integrating enterprise applications as manageable entities. It allows users to extend their enterprise management tools directly to applications creating a comprehensive end-to-end management capability that includes measuring application availability, application performance, application usage, and end-to-end transaction response time. To date ARM 3.0 specifies an interface for the Java programming language. Applications use ARM by creating objects that implement the interfaces in the org.opengroup.arm3.application package. The ARM C-binding has been specified in ARM version 2.0 since November 1997 and adopted by The Open Group in 1998. The AIC standard also requires software being modified with its API calls.

The use of these standards (ARM and AIC) does not necessarily ensure a better application management solution. In addition, instrumentation is usually not an easy task without source code modification and recompilation. The value is likely determined by the use of the standards, i.e. it is more likely based upon the level of analysis done to determine how to apply these techniques, either through plug-ins, source code modification, or dynamic instrumentation, to various applications.

3. Building Sensors for Applications

Unlike physical system resources that are finite, the number of possible application architectures, configurations, and interfaces for applications is almost unlimited. The task of application management becomes more complex as many interdependent applications are often required to fulfill a business function such as web banking and B-to-B web applications. Historically, there is little involvement in application itself for system management. Manageability is often a low development priority or an after thought for software development, and it may have different requirements for different running applications.

To use CIMOM for managed resources, we need CIMOM provider implementations. The Apache web server is used as an example to illustrate the needed steps to build sensors and gather performance data from applications. The first step in developing a CIMOM ARM Provider for an application is to understand the application for possible instrumentation. For Apache web server we develop a plug-in to capture the entry and exit points of a web request with minimum overhead (roughly 1 micro-second per record) and without modifying any server source code. Each point in time is recorded in a trace file, along with related information such as the requested URL and the process ID/thread ID of the web process. A Java-based visualization tool, RUME Gantt Chart, is developed and used to visualize such trace files. Figure 3 shows 10 web processes handling web requests along the timeline. There are 10 URLs in Figure 3, each of which got 100 hits.

Each timeline in Figure 3 represents an Apache server process. Rectangles along a timeline indicate web requests for various URLs, including static HTML pages and CGI requests, and the width of a rectangle represents the elapsed time for that request. Different URLs are represented by different colors for easy visualization, and black areas indicate periods when a server process is not serving any requests.

The server plug-in is then modified and linked with an ARM library to generate performance data. Note that binary CGI programs and many legacy software applications may not support dynamically loadable modules or plug-ins. In these cases an alternate approach is to use dynamic instrumentation techniques such as those in the DPCL library [8] or DynInst [9] to add the needed calls for application management.

As a standard, the ARM C-binding API is made up of a set of function calls, including `arm_init()`, `arm_start()`, and `arm_stop()`. By developing an ARM library and linking the server plug-in with this library, we are able to capture transaction response times (i.e. elapsed times of individual URL requests) of the Apache web server. The statistics is obtained live from the running Apache server, thus provides us the first known ARM instrumentation with live system management data without modifying or recompiling the Apache server itself.

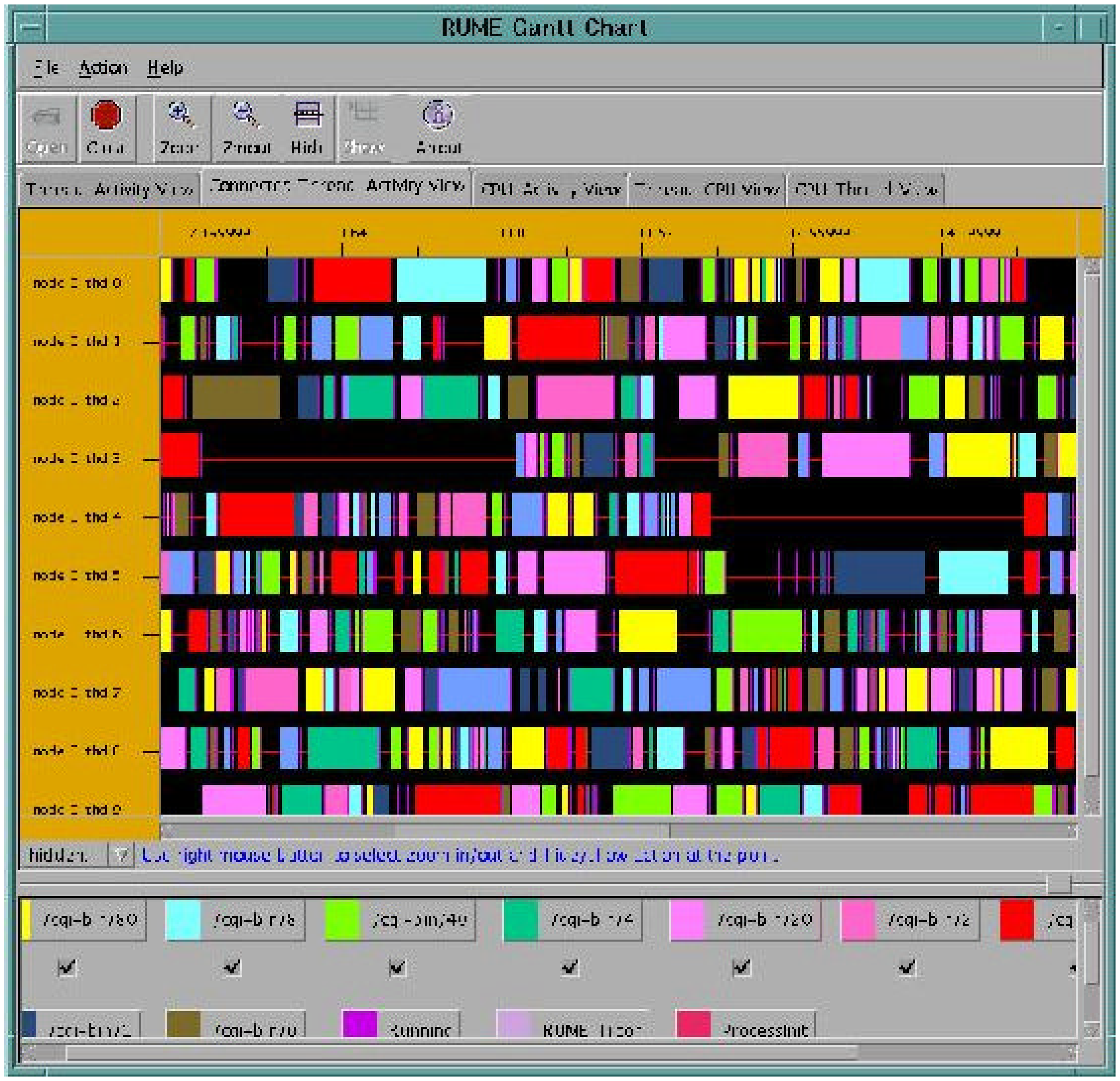


Figure 3. Gantt Chart visualization for 10 web server processes handling requests

An ARM agent is also implemented to collect the statistics from multiple applications and accept requests for stored performance data. It collects application statistics and accepts requests via a certain IPC socket. A CIMOM ARM Provider is developed for CIMOM using its provider interface to retrieve performance data from the ARM agent. Figure 4 shows the interaction between the ARM Provider, ARM Agent, and Applications that are instrumented with ARM APIs.

The ARM agent handles multiple applications concurrently to minimize the impact on system resources, and it can be run in another machine if the server machine is overloaded. Note that there is nothing prevents its client from requesting performance data directly, i.e. clients of the ARM agent can communicate directly with the ARM agent. On the other hand, CIMOM and its Provider interface provide an open standard for management data. The CIMOM ARM Provider is built as a client of the ARM agent to retrieve application statistics for performance monitoring. Thus, CIMOM clients will be able to acquire application statistics through the ARM provider.

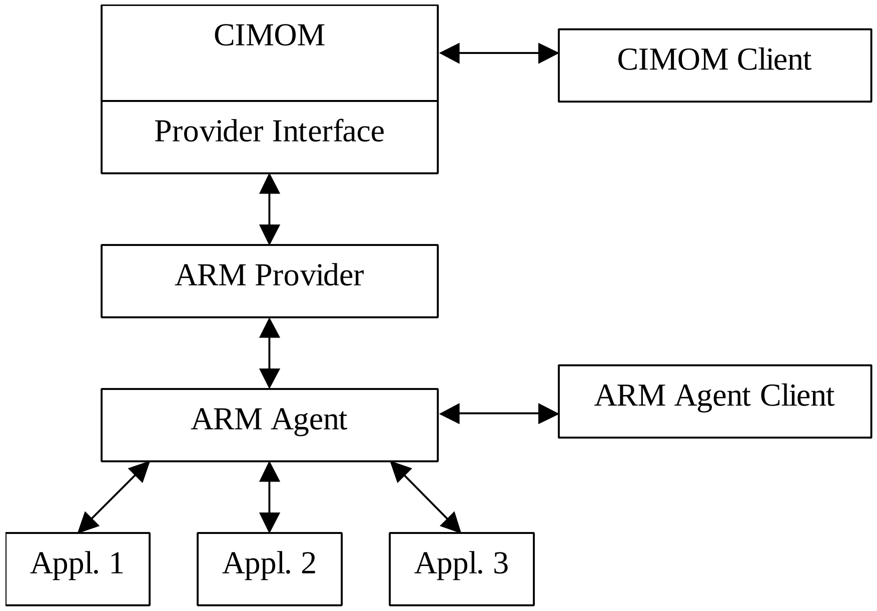
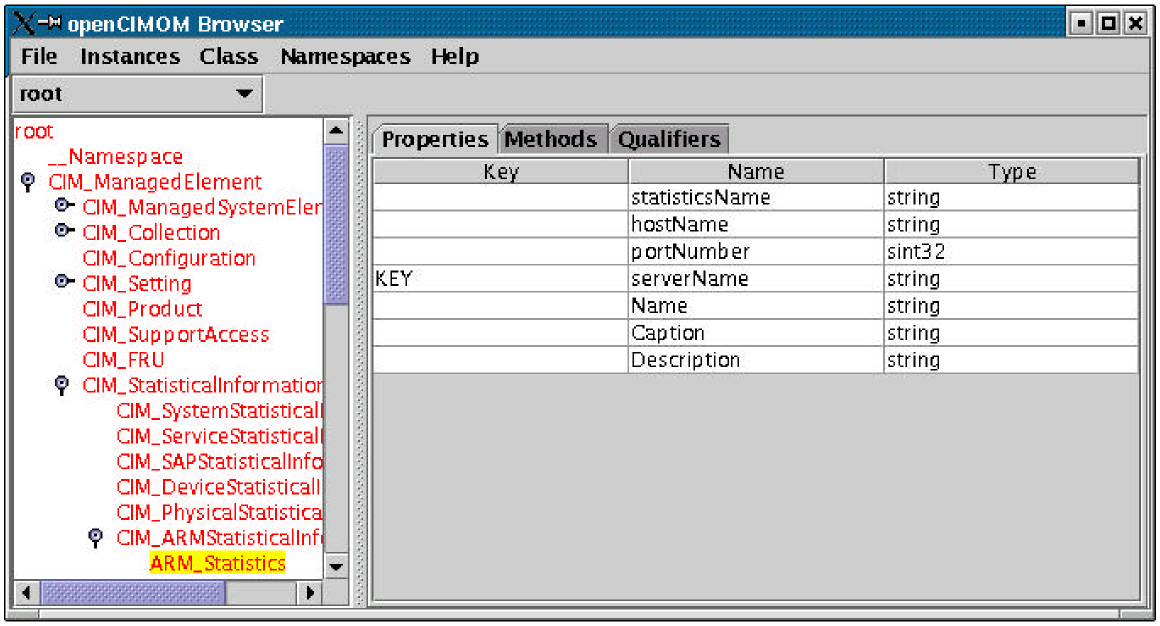


Figure 4. One ARM agent handles multiple instrumented applications

Figure 5 shows the properties of the CIMOM ARM Provider using the openCIMOM browser of the SNIA CIMOM. It can be seen that the ARM_Statistics Provider is derived from CIM_ARMStatisticalInformation, which is derived from CIM_StatisticalInformation. The properties of the provider include the host name and port number where the ARM agent is running, the name of the ARM agent, and the name of the statistics which its client is currently interested in.

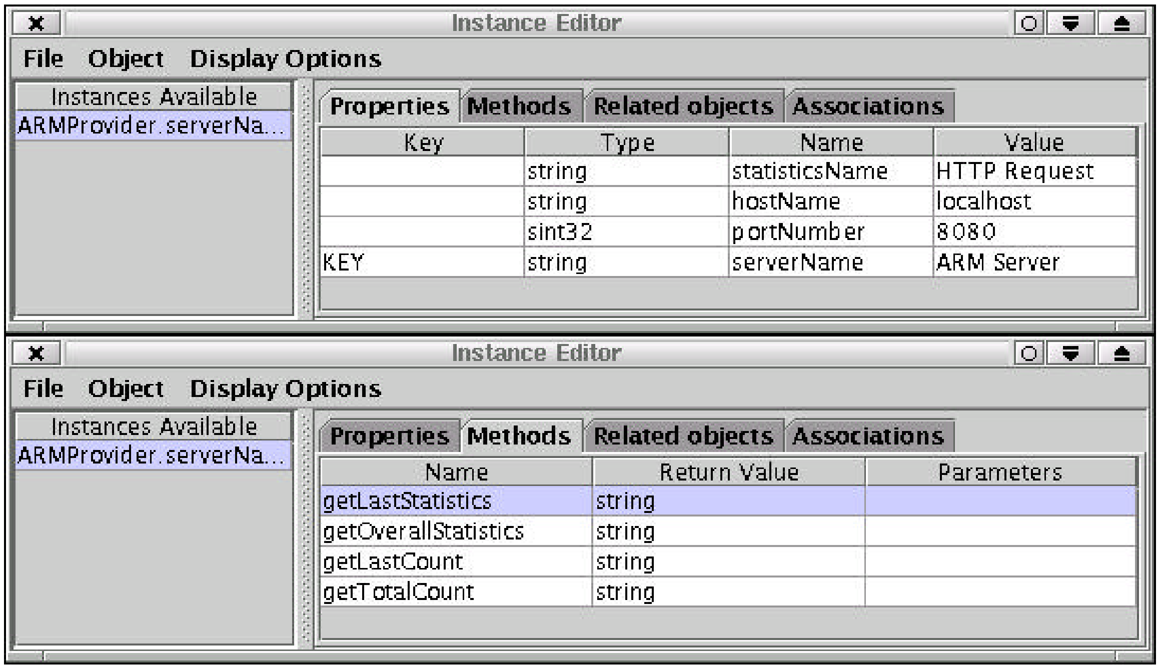

openCIMOM Browser
File Instances Class Namespaces Help
root
Properties Methods Qualifiers

root
- __Namespace
- CIM_ManagedElement
  - CIM_ManagedSystemEler
  - CIM_Collection
  - CIM_Configuration
  - CIM_Setting
  - CIM_Product
  - CIM_SupportAccess
  - CIM_FRU
  - CIM_StatisticalInformatior
    - CIM_SystemStatisticalI
    - CIM_ServiceStatisticalI
    - CIM_SAPStatisticalInfo
    - CIM_DeviceStatisticalI
    - CIM_PhysicalStatistica
    - CIM_ARMStatisticalInf
      - ARM_Statistics

Key	Name	Type
	statisticsName	string
	hostName	string
	portNumber	sint32
KEY	serverName	string
	Name	string
	Caption	string
	Description	string

Figure 5. An ARM Provider for SNIA CIMOM

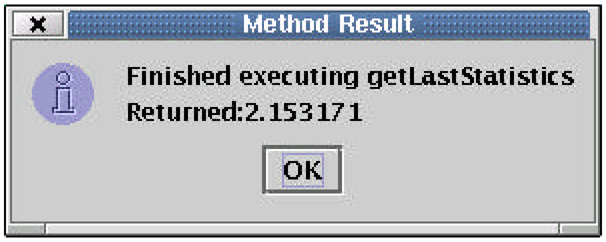
Figure 6 shows a set of property values and methods of the ARM Provider. URL statistics could be individually collected and distributed by the ARM agent, or they could be aggregated as a single entity with the name "HTTP Request". As a result, system management data could be aggregated or based on individual URLs. The latter could be useful for projects such as the Heterogeneous Workload Management [10] or SLA-based projects, which require URL-based performance statistics for management actions. Figure 7 shows the result returned from the getLastStatistics() method of the ARM Provider for the statistics "HTTP Request".


Instance Editor
File Object Display Options
Instances Available
ARMProvider.serverNa...
Properties Methods Related objects Associations

Key	Type	Name	Value
	string	statisticsName	HTTP Request
	string	hostName	localhost
	sint32	portNumber	8080
KEY	string	serverName	ARM Server

Name	Return Value	Parameters
getLastStatistics	string	
getOverallStatistics	string	
getLastCount	string	
getTotalCount	string	

Figure 6. Properties and methods for the ARM Provider



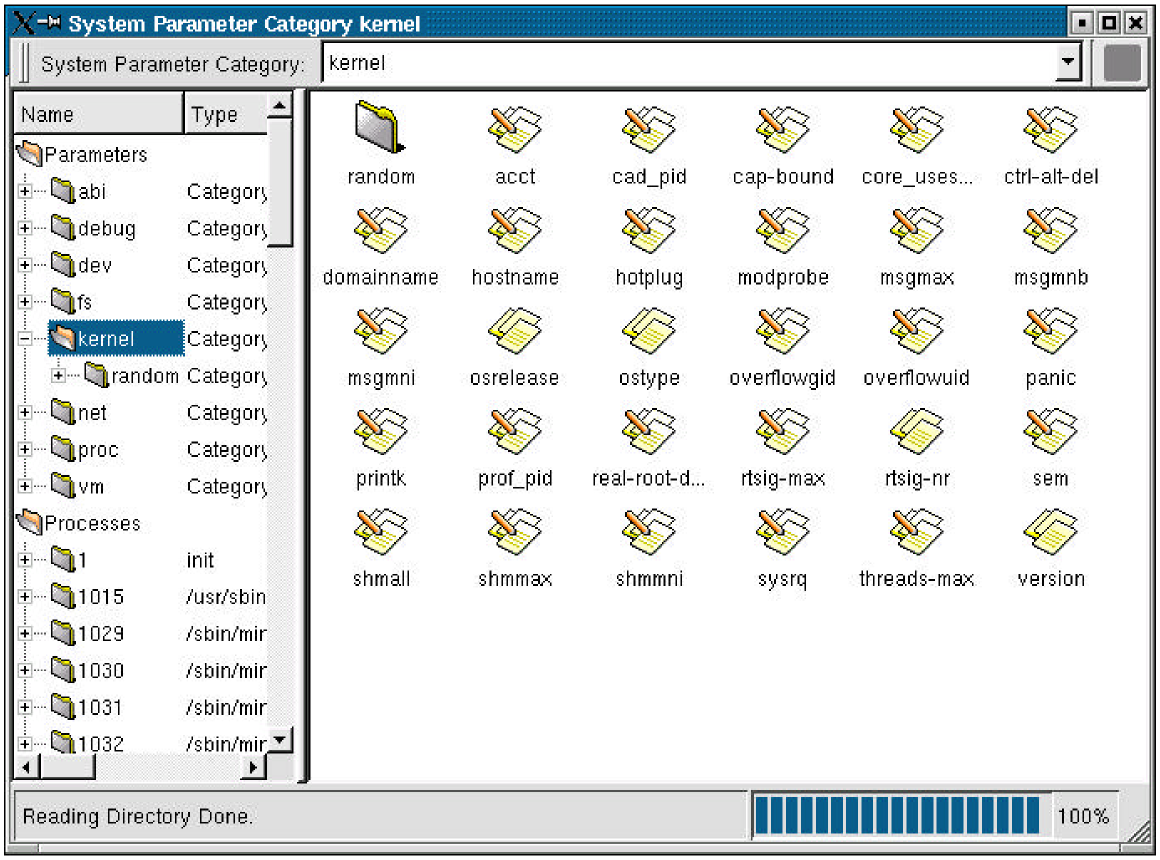
Finished executing getLastStatistics
Returned:2.153171

Figure 7. Execution result from the getLastStatistics() method of the ARM Provider

While the ARM API is specified and supported by The Open Group, the communication protocol between the ARM Agent and its clients is totally up to the ARM agent designer. The design of the ARM agent is critical to efficiently collect various statistics from multiple instrumented applications. Typically a web server includes multiple applications, such as the Apache web server, Application server, backend database, and programs using the Common Gateway Interface (CGI) [11]. How to efficiently monitor these applications and provide system management tools with monitoring data is critical for the success of the ARM Provider.

4. Expose Tunable Parameters and Build Actuators

Actuators are control knobs that can be used to change behaviors of system resources. As important as an infrastructure for application management, operating system parameters, especially tunable system parameters, are critical actuators for enterprise Linux systems. Such tunable system parameters include maximum number of open files, maximum size of shared memories, maximum map count, maximum number of threads, etc. In addition to these system-wide parameters, per-process parameters such as the maximum number of open files and maximum number of child processes are equally important to individual processes. Many applications and system services also have critical parameters in configuration files that provide specific control for various options. For example, the Apache web server uses the httpd.conf configuration file for specifying parameters such as the number of server processes at the starting point, the minimum and maximum number of spare server processes, among many other things. It is our belief that a good system management environment should provide control knobs and user-friendly tools to cover these basic areas.



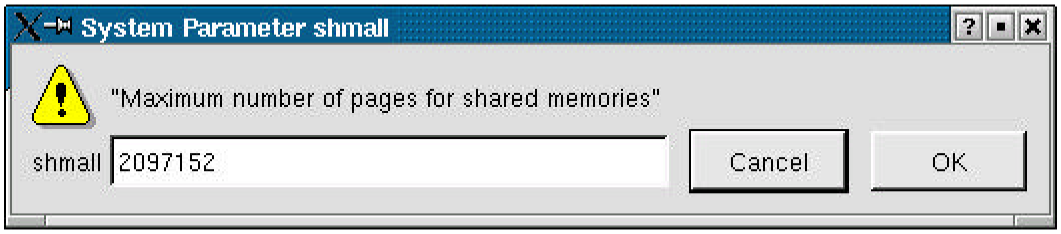
Name	Type
Parameters	
abi	Category
debug	Category
dev	Category
fs	Category
kernel	Category
random	Category
net	Category
proc	Category
vm	Category
Processes	
1	init
1015	/usr/sbin
1029	/sbin/mir
1030	/sbin/mir
1031	/sbin/mir
1032	/sbin/mir

Reading Directory Done.

Figure 8. A user-friendly interface for tunable system parameters

4.1 Tunable System Parameters

Tunable system parameters of recent Linux kernels (2.1 and later) can be found in the /proc/sys directory, which is in a special, software-created file system that is used by the kernel to modify kernel parameters as well as export information. For these tunable system parameters we develop the kparam system management tool to provide a consistent and user-friendly interface. The management tool works for all Linux systems with the /proc/sys interface. Figure 8 shows its user-friendly GUI for tunable system parameters running on a Linux system. A file icon with a pen indicates that the tunable system parameter is modifiable, while a file icon without a pen indicates that the parameter is merely readable. Double clicking on a file icon brings up a dialog, shown in Figure 9, which allows the user with appropriate privilege to change the value of the tunable system parameter.

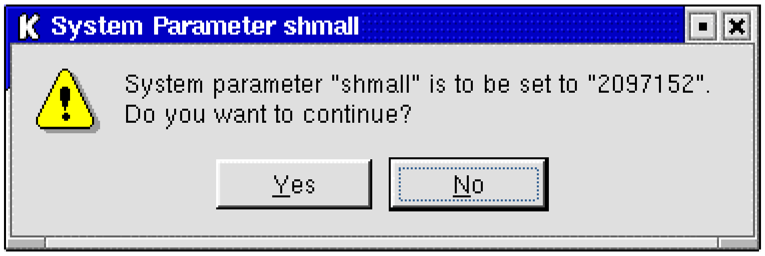

System Parameter shmall

"Maximum number of pages for shared memories"

shmall 2097152

.
Figure 9. A popup window for modifying system parameter “shmall”

Figure 9 shows the popup window for modifying the “shmall” system parameter, which controls the maximum number of total pages for shared memory segments. The quoted string shown in Figure 9 can be easily modified or replaced. All help strings and mini menus are stored in a configuration file for the management tool. This simplifies the addition of help strings for extra entries and also makes replacement easy. Double clicking on a readable file icon brings up a similar popup window for confirmation. Users without appropriate privilege will find a parameter is merely readable, while users with root privilege will find most parameters modifiable. After a parameter is modified in the popup window, the user clicks the OK button and brings up another popup window to confirm or reject the modification, shown in Figure 10.


System Parameter shmall

System parameter "shmall" is to be set to "2097152".
Do you want to continue?

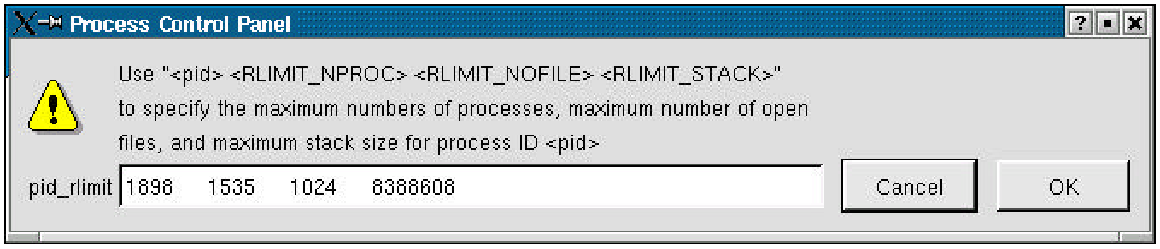
Figure 10. A popup window to confirm or reject changes

The popup window to confirm or reject current modification gives the user a second chance before committing the change. This is because changes for tunable system parameters take effect immediately and must be handled carefully.

4.2 Per-Process Parameters

Per-process parameters in most Linux/Unix systems are captured in a per-process data structure called "rlimit". Typically these parameters such as the maximum number of child processes and maximum number of open files for a process are accessed through a pair of system calls, getrlimit() and setrlimit() while running inside the process. A kernel module is developed and is loaded automatically by the management tool to modify per-process parameters from outside the process. It creates a file entry pid_rlimit in the /proc/sysman directory. All reads and writes to the file entry are reflected immediately in the specified process. Given a process ID and new values for rlimit through the file entry, the kernel module locks the task list (i.e. the process list) for read and finds the process before unlocking the task list. It then checks the rlimit in the data structure of the task and makes necessary modifications.

Figure 11 shows a popup window for modifying per-process parameters. In this figure the process whose process ID is 1898 currently could have up to 1535 child processes, up to 1024 open files, and up to 8 Mbytes stack size. These limits can be changed easily by entering different numbers in this popup window, using exactly the same GUI as for tunable system parameters.



Use "<pid> <RLIMIT_NPROC> <RLIMIT_NOFILE> <RLIMIT_STACK>"
to specify the maximum numbers of processes, maximum number of open
files, and maximum stack size for process ID <pid>

pid_rlimit 1898   1535   1024   8388608

Figure 11. A popup window for modifying per-process parameters

The /proc file system contains a directory for each process running on the Linux system. Since the same GUI is used for processes, file entries in these process directories such as environ for environment variables can be modified as well.

4.3 Application Configurations

Many applications use configuration files to specify various options. Entries in configuration files may include control knobs and useful hints for an application or its service. Changing an entry in a configuration file may change an application or its service completely. Traditionally many Linux/Unix services are provided through configuration files, such as /etc/passwd for user accounts, /etc/fstab for file systems, httpd.conf for Apache, etc.

There is no doubt that configuration files are part of the actuators for system management [12, 13], and there have been various noble attempts to ease the problem of configuration files. Some popular packages have GUI front-ends built for them, but each front-end works differently. Other packages have a range of interfaces that can be used, from a raw text file to a text-based front-end. Another approach is to custom build a consistent set of front-ends for a number of software configurations, as is done with the KDE control panels and the Linuxconf project.

It is relatively simple to use configuration files for applications and services alike, however, there is no standard for other programs to find out what options a program or service has and how to set those options (potential actuators for system manageability). On the other hand, we have just seen in the last section how tunable system and per-process parameters are managed with the simple yet powerful GUI. Thus, we extend the kernel module to build actuators in /proc/config, a virtualized directory, as a consistent interface for user accounts, Apache setup, among many others.

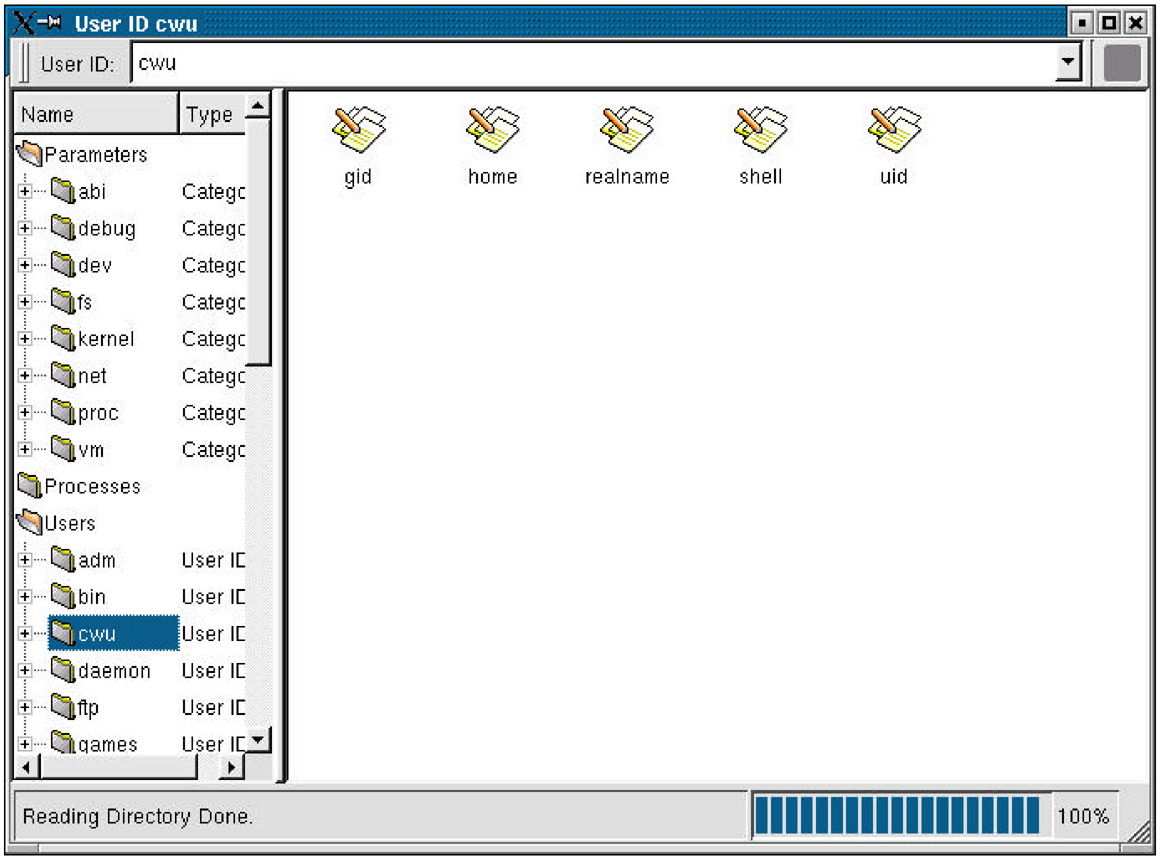


Figure 12. A number of icons representing fields for user “cwu” in /etc/passwd

The kernel module is firstly extended for user accounts to show its novelty and simplicity. With a couple hundred lines it captures all fields in /etc/passwd into subdirectories in /proc/config/acct, one for each user account. Figure 12 shows five icons for the user “cwu” in /etc/passwd. Double clicking on an icon, say the “realname” icon, will bring up a popup window for possible modification of that particular field. Figure 13 shows the popup window displaying the real name in life for the user “cwu” in /etc/passwd.

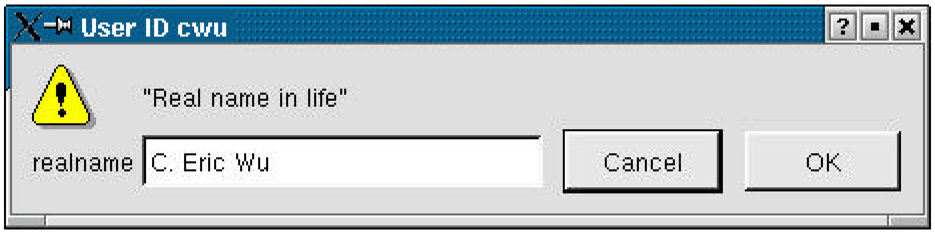


Figure 13. A popup window displaying the real name in life for user “cwu”

This example shows that applications could keep their configuration files while it is also feasible to create a centralized place for local configuration information and a consistent approach for extracting and modifying their various options. In this way, one would be able to browse the hierarchy of configuration information using simple shell commands or the browsers of his or her own choice. A reasonable next step is to develop a few more examples, such as capturing Apache configuration elements in /proc/config/apache as actuators for the foundation of adaptive resource management.

The management tool is built using Qt, a cross-platform C++ GUI application framework and one of the most popular graphic libraries in Linux. KDE was also built using Qt. The tool monitors file entries in three areas: kernel parameters in /proc/sys, processes in /proc and /proc/sysman/pid_rlimit, and entries created in /proc/config by its kernel module. All these parameters, including the per-process ones, are defined in file entries that can be browsed and modified using simple shell commands such as “cat” or “echo”. Thus, there is a clear separation between control knobs and means for modification. The management tool is basically a graphical interface that browses over these file entries and provides popup windows for modifying them, while the kernel module provides actuators for per-process parameters and entries in configuration files.

5. Summary and Future Work

System resource management, including both management and manageability, is a broad and important area for enterprise Linux. The ultimate goal of system management is autonomic computing, in which management software collects management information from managed resources and changes system resource allocation and services adaptively. Manageability, including the ability to create sensors and actuators as examples given in this paper, provides the needed elements and control for adaptive system management.

The goal of adaptive system management is to change application and/or system configurations based on changing environments. In this paper we build sensors and actuators, the foundation for adaptive resource management. We expose tunable system parameters through a consistent and user-friendly GUI, and create actuators for per-process parameters and entries in configuration files. For sensors we build the CIMOM ARM infrastructure to monitor application performance. The CIMOM ARM infrastructure, consisting of a standalone ARM library, an ARM provider, an ARM

Agent, and a sample web server plug-in shows a promising approach for system management in the area of application run-time. We believe that this is one of the most critical areas in enterprise Linux for the years to come. As CIM/CIMOM becomes more and more popular, we expect more computer vendors build CIMOM providers for various resources.

Our future work includes creating actuators for Apache web server using its configuration file, and adaptive algorithms that use these sensors and control knobs to form the control loop for adaptive resource management.

References

1. Autonomic Computing: IBM's Perspective on the State of Information Technology, Paul Horn, http://researchweb.watson.ibm.com/autonomic/manifesto
2. Common Information Model: Implementing the Object Model for Enterprise Management, W. Bumpus et. al., Wiley Computer Publishing, ISBN 0-471-35342-6.
3. Pegasus Developers and Users Manual, The Open Group, May 2001, available at http://www.opengroup.org.
4. SNIA CIM Object Manager: Architecture and Developers Guide, version 0.1, available at http://www.snia.org.
5. Application Response Measurement 2.0 API Guide, Hewlett-Packard Co. and Tivoli Systems, Inc., November 1997.
6. Application Response Measurement, Issue 3.0, Open Group Technical Specification, March 2001.
7. Application Instrumentation and Control (AIC) API, Open Group Technical Standard, November 1999.
8. Dynamic Probe Class Library Class Reference, IBM Publication SA22-7241, also http://www.ptools.org/projects/dpcl.
9. DYNINST: An Application Program Interface for Runtime Code Generation, http://www.dyninst.org.
10. Adaptive Algorithms for Managing a Distributed Data Processing Workload, J. Aman, C Eilert, D. Emmes, P. Yocom, and D. Dillenberger, IBM System Journal, Vol. 36, No. 2, 1997.
11. CGI: Common Gateway Interface, please see http://www.w3.org/CGI and http://hoohoo.ncsa.uiuc.edu/cgi.
12. Systems Management Roadmap for Linux, Adrian Schuur, October 2001, at http://w3.opensource.ibm.com/pub/ibmcimom/LinuxSystemsManagementRoadmap_101201.prz.
13. How to fix the Unix Configuration Nightmare, Matthew Arnison et. al., February 2002, at http://freshmeat.net/articles/view/400.