RC22972 (W0311-039) November 5, 2003
Computer Science

IBM Research Report

Resource Management and User Registration in an eUtility

Barry Leiba, Marion Blount, Wolfgang Segmuller
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Resource Management and User Registration in an eUtility

Authors

Barry Leiba
Marion Blount
Wolfgang Segmuller

Abstract

A service provider offering an eUtility faces a number of administrative challenges in deploying that service. We describe a research prototype that addresses two challenging areas in such a deployment. First, when a service provider offers an eUtility to enterprises with existing user bases, the process of registering those existing users, as well as the process of managing user registrations as people come and go in the enterprise, can be cumbersome and error-prone. Our system takes advantage of the enterprise's current user-management tools to ease user-registration management for the eUtility. Existing users can easily be signed up for the eUtility service on organizational boundaries, as the service is deployed in different parts of the enterprise.

Then, as the eUtility is used to serve many customers, the provider will find itself coping with varying needs and expectations from the different customers. To avoid dedicating resources statically, it will need tools to monitor usage and to adjust resource allocations according to those measurements as compared with the service-level agreements for the different customers. Our prototype performs those monitor and control functions, allowing the service provider to make those allocation adjustments automatically.

Introduction

In 2000, as IBM's interest in the concept of eUtilities grew, the Internet Messaging Technology group in IBM's Research Division set out to build a prototype eUtility to explore the feasibility of providing services to multiple customers in a new way. The service we chose as the basis for the prototype was e-mail notification, since that gave a useful function that we could actually deploy to a group of test users, it made use of e-mail, filtering, and notification technology that we already had developed,[1,2] and it presented the challenge of tying the eUtility service into an existing service that our customer might already be providing for itself.

The notification service would inspect incoming mail for registered users, would filter that mail using criteria specified by the user, and would notify the user about mail that passed the filters – mail that the user, by setting up the filtering criteria, has categorized as "important". Each user could have a set of notification "devices", such as mobile phones and instant-messaging names, which would be used to send the notifications to the user.

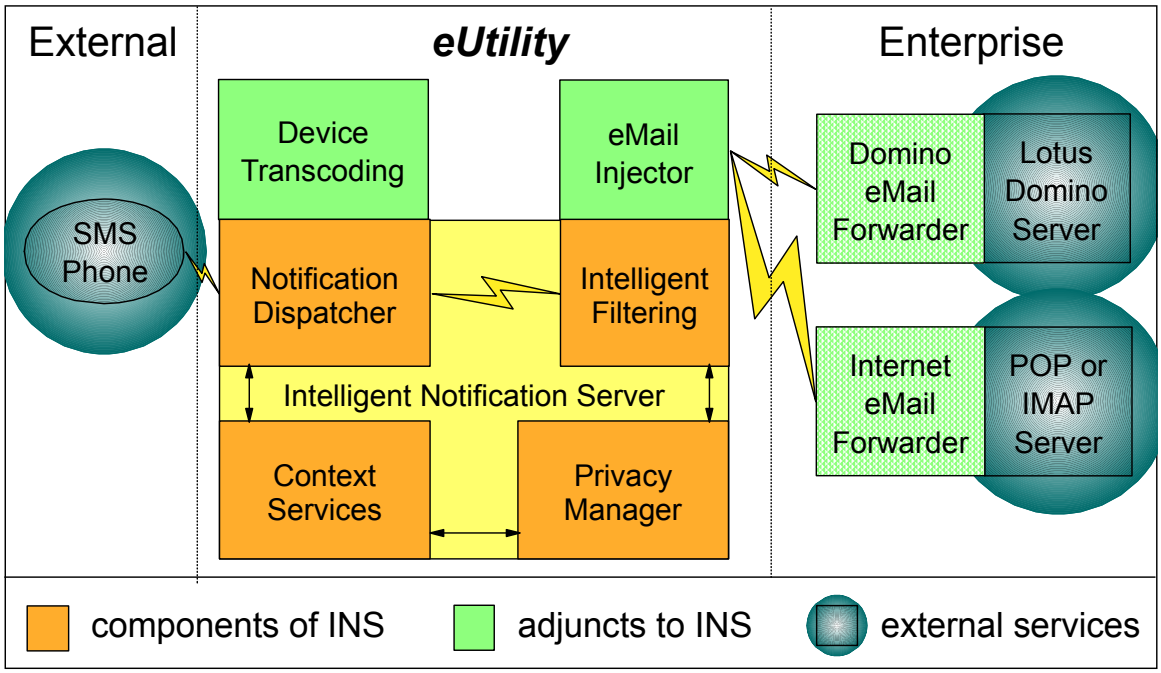
When an enterprise would sign up for this service, we would have to start inspecting and filtering mail for a registered set of that enterprise's users. A new customer might want to start with a limited set of users as a trial, and then might want to sign users up in groups over time, perhaps on organizational boundaries.[3] This would have to be easily done, and ought to be done by an administrator of the customer's choosing, not by the service provider. That is, we should provide a way for the customer itself to handle registration of its own users, and that mechanism would have to be flexible enough to work for many different customers.

Once the service was running and had multiple customers, the service-provider might be required to meet different Service-Level Agreements (SLAs) for each customer.[4] During periods of peak activity, the service-provider would have to ensure that it provide service as promised, and, if resources become overcommitted, that it give preference to those customers with the strictest SLAs.

Building the Service

The first step was to create the service that we would offer. Using our Internet Messaging Framework toolkit,[1] we created modules that would use the Internet Message Access Protocol (IMAP)[5] or Lotus Notes native facilities to read a user's new mail and put it through the filters (for this prototype, we set up some "standard" filters; setting up a user interface to allow users to define filtering criteria was beyond the scope of our project). We set up the Intelligent Notification Server (INS)[2] to perform the notifications. These two ends were tied together by writing triggers by which the filters would tell INS to make the notifications. A block diagram of the service design is shown in ***Figure 1***.

Figure 1 – The Notification eUtility



A test of the system showed that it worked: we could send "urgent" e-mail to a user and have an SMS message (containing the e-mail's sender and subject) sent to the user's mobile phone. Now we had to make an eUtility out of it.

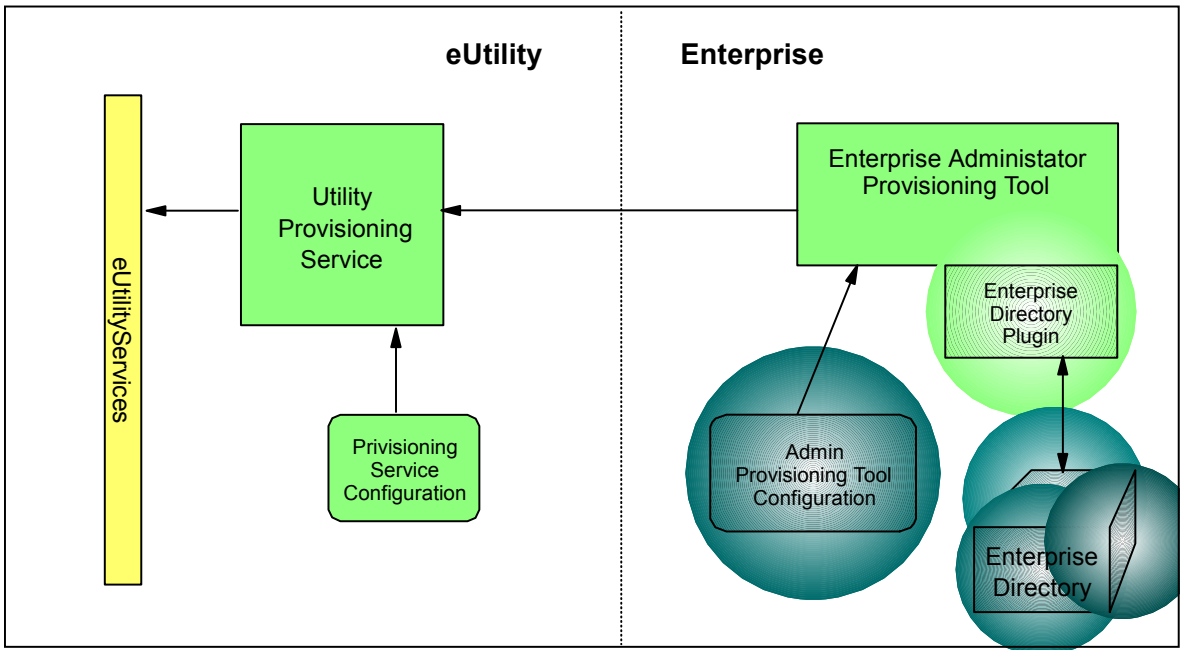
User Registration and Provisioning

We designed a registration tool with interchangeable interfaces, so we could plug into different directory services on the customer side, and different registries on the eUtility side. For this prototype, we implemented a configurable LDAP (Lightweight Directory Access Protocol)[6,7] directory service, and a registration service that could contact a servlet at the eUtility to register users and devices with INS. The registration tool would read the enterprise directory using LDAP, would look for particular (configurable) fields that would provide users' mobile phone numbers and instant-messaging IDs, and would allow an administrator to choose which users to register with the service. Because we anticipate that staged registration is likely to be done on organizational boundaries (sign up the Data Processing department, for instance), the tool is able to traverse the enterprise's organizational hierarchy if fields are available in their LDAP directory to do that.

Of course, while we could make the tool general, some customization would be needed to deal with each enterprise's specific LDAP directory. Specifically, we have to map the data elements we need to LDAP field names, we have to define the LDAP searches needed to perform the tool's functions (find a user, traverse the hierarchy both up and down, etc.), and we have to tell the tool's user interface what LDAP fields to show the administrator so that display makes sense. If there is a "typical" setup, we can provide a typical configuration file, which maps LDAP fields and sets up LDAP searches in a typical way. Otherwise, an enterprise administrator would have to make changes to the

configuration file suitable to the enterprise's LDAP directory. ***Figure 2*** shows a block diagram of the registration tool and associated services and components.

Figure 2 – Provisioning the eUtility



The functions we chose, after experimentation and use, are as follows (the actual names for the functions, as they appear in the interface, are configurable):

- Expand one level below this user.
- Expand all levels below this user.
- Go one level up from this user.
- Go all levels up from this user, to the top of the hierarchy.
- Find this user's administrative assistant.
- Find all users this assistant supports.
- Select (and de-select) this user for registration.
- Select (and de-select) this user and one level below.
- Select (and de-select) this user and all levels below.
- Show this user's entire (formatted) LDAP entry.

That last function is useful for selecting the correct "John Smith" in case there is a doubt. The tool's user interface also makes it easy to search for specific users, and to toggle back and forth between the hierarchical list and a simple list of all selected users. Users may be selected separately from different parts of the organization, and the selections will be collected. When the selections are made, the administrator will click a "register now" button, and the selected users will be processed.

As a test of the configurability, we have configured the tool to work with three different LDAP directories, all with different layouts and fields. We have also configured the other end of the tool to talk to different "registration" systems, and we have a configuration that copies the selected users' e-mail addresses to the clipboard, so they can

then be pasted into the "to" field of a new message – some of us are using this today as our normal way of searching our corporate directory.

In the following screen captures, we show an example of registering everyone who works for "Bill Jackson"; first (***Figure 3***, ***Figure 4***) we search for all names matching "Jackson, B"…

Figure 3 – The Registration Tool, an Example

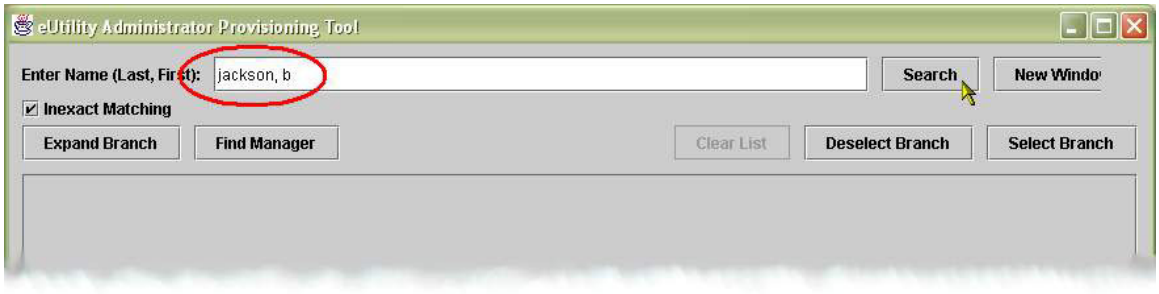
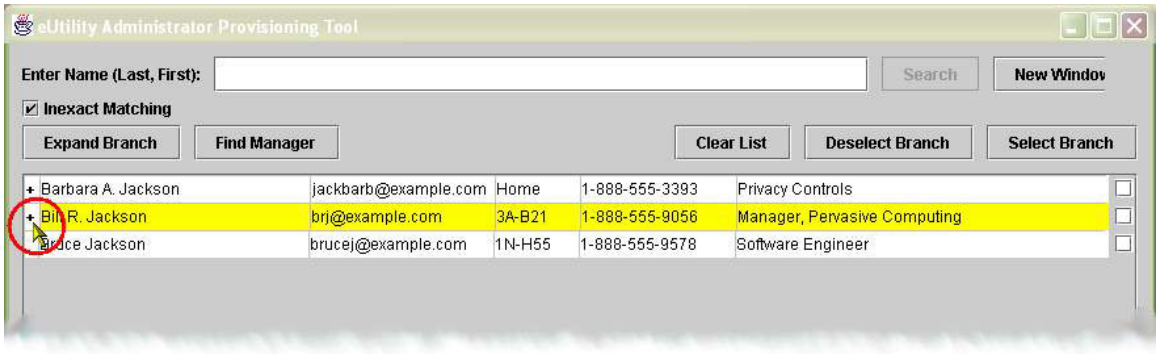
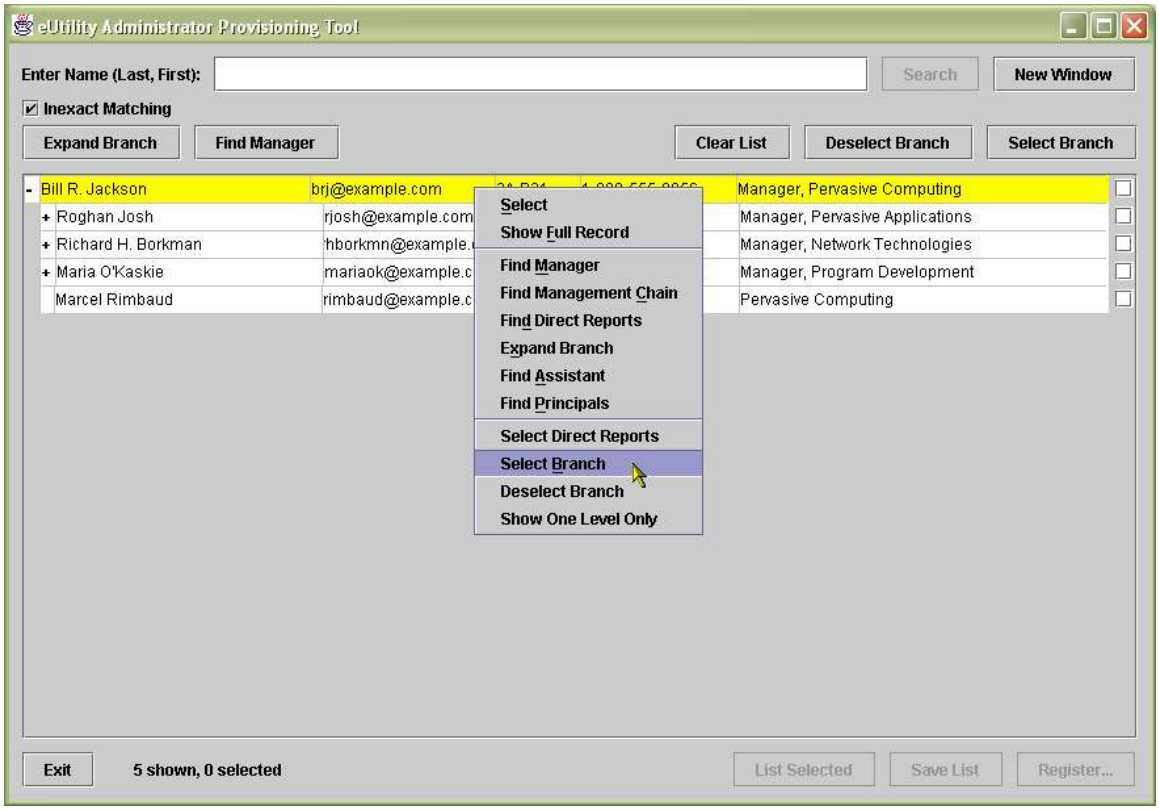


Figure 4



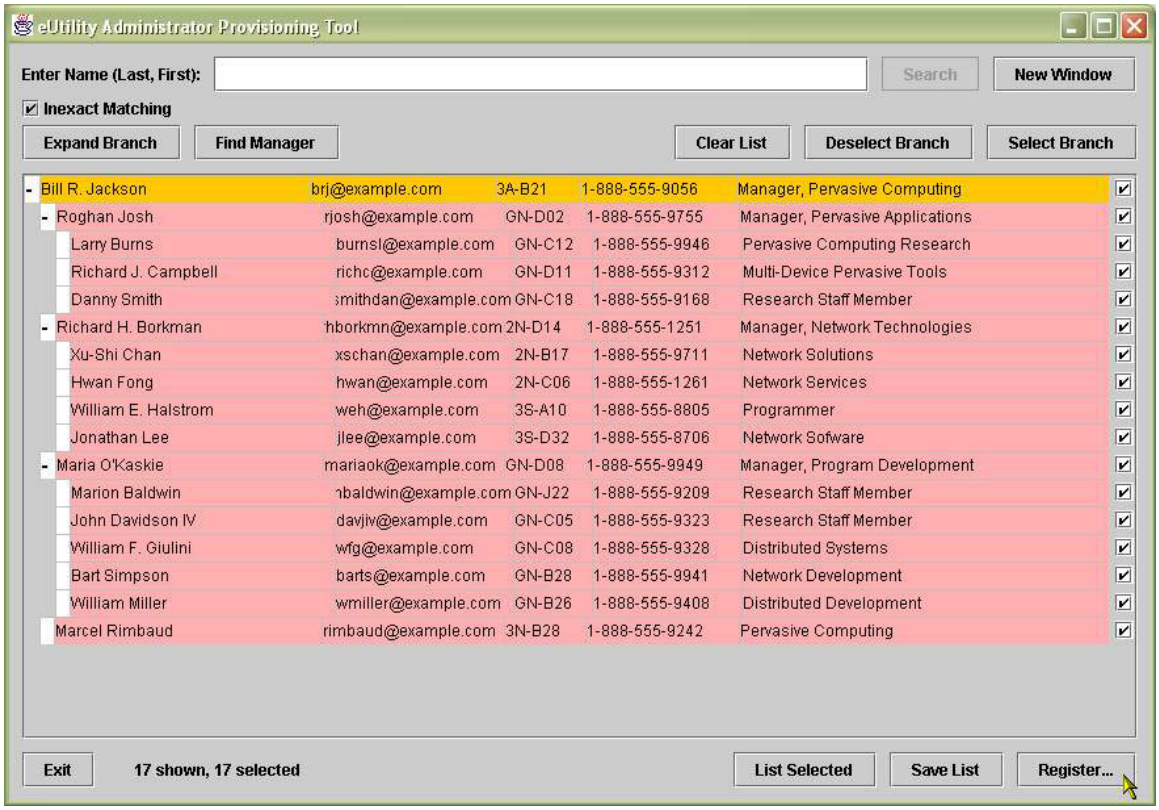
…then (***Figure 5***) we expand one level below Bill (perhaps the administrator is just making sure that's the group he wants), by clicking on the "+" symbol that indicates something to expand (Bill is a manager)…

Figure 5



…and then (***Figure 6***) we use "select branch" to choose everyone in that part of the organization (there's both a button and a pop-up menu item for this; here we show the pop-up menu).

Figure 6



At this point, Bill Jackson and everyone who reports through him are selected for registration. We can now do more searches and navigation, select more people, turn off the selection on some if that's what we want (by turning off the check box on the right), or click "Register…" to process the registrations now.

Enterprise Security & Privacy Implications

The registration tool and its process of tying the enterprise customer's employee directory into the eUtility provisioning is very useful in making the user registration process more efficient and less error-prone, but does it expose the customer's internal systems to security risks and invasion of privacy by allowing such access? We believe it does not: the registration tool is designed to be used by administrators of the customer's own choosing, usually by employees who already have access to the information anyway. Of course, the utility-provider may offer to do this administration for them, as an optional service, in which case the customers who choose that must be aware that they are exposing information about their employees and their business hierarchy to the service provider. The scope of the exposure is limited by the flexible configuration of the tool; only the information needed to do useful searches and to select users for registration will be retrieved, displayed, or sent to the registration server. If a user's e-mail address is needed, but not her telephone number, then the tool may be configured so that it never retrieves the telephone number. LDAP authentication credentials are also hidden from the administrator through the configuration, and the tool may be run as a servlet, giving

the administrator no access at all to the configuration parameters or to the communication path between the registration tool and the directory server.

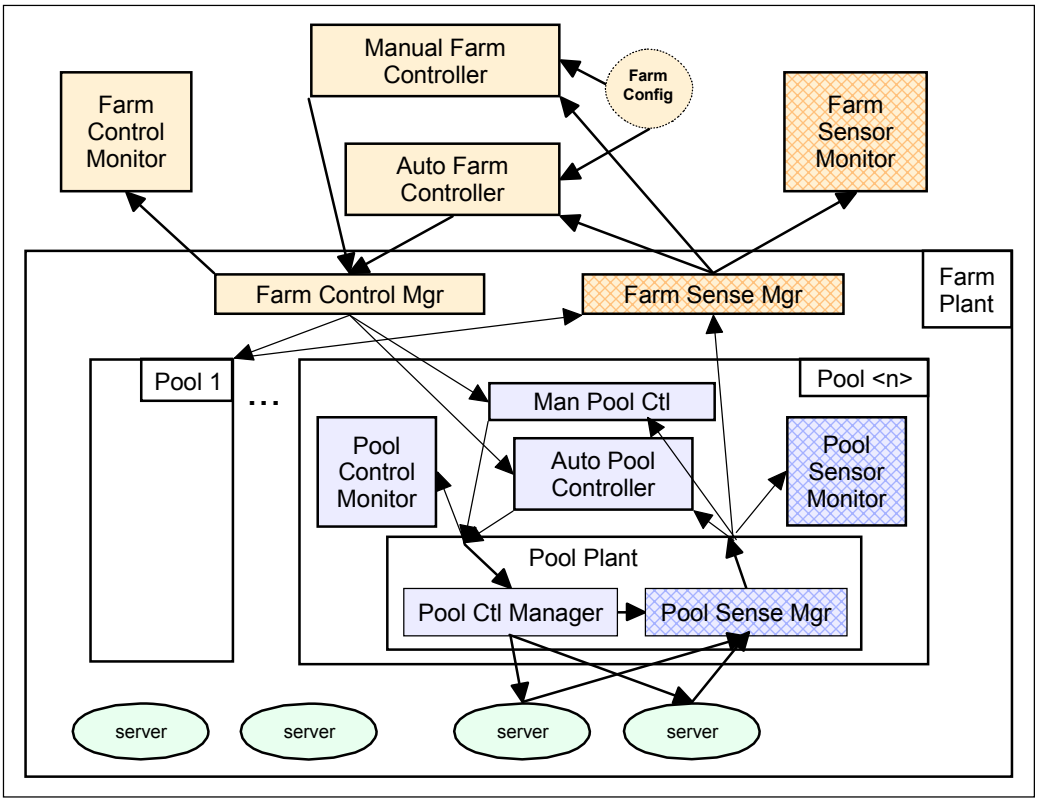
Resource Monitoring

Now that users are registered and the eUtility is running, and as we get a number of customers signed up, we need to monitor the eUtility's workload and performance, report on peaks, bottlenecks, and over- or under-committed resources. We would also like to make resource adjustments automatically, responding on demand to changes in workload. To accomplish this we designed a system for monitoring a "farm" of servers, in which each server may be placed in one of the farm's "pools". Servers would be put in pools and pools given work from customers based on those customers' Service-Level Agreements and the effect that the load on the farm has on whether all SLAs can be met.

The monitoring system comprises a set of java classes and components that implement those programming interfaces, along with user interfaces to give eUtility administrators feedback and easy ways to set and adjust the controls on the system. A Farm Controller provides overall monitoring and control of the farm. Each server pool has a Pool Controller that monitors that pool and reports to the Farm Controller. The control system defines a listener interface that each controller implements, and the Pool Controller registers its listener with each server that it's monitoring, so that it may receive operational data from that server and the services running there.

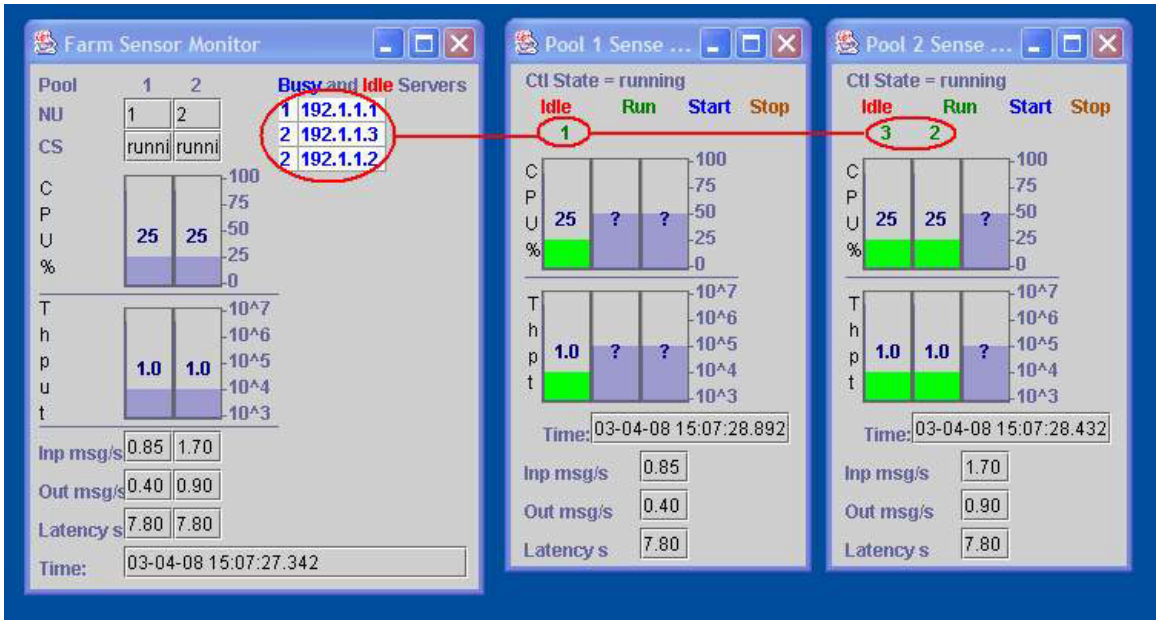
We monitor each server's overall workload using CpuListener and CpuDetailsListener interfaces. Each service is monitored for throughput using a ThroughputListener interface, giving us information that may be used to balance the workload among multiple services that share a single server. The pool collects information from all the servers. A block diagram of the monitor & control system is shown in ***Figure 7***. On the control side, the farm and each pool have Control Monitors and both automatic and manual Controllers. Control decisions are made by the Controllers, getting feedback from the Control Monitors. There are Control Managers for each component, which are responsible for conveying the control signals from the Controllers and executing them. On the monitoring side, the farm and each pool have Sense Managers and Sensor Monitors, which collect information and feed it up and into the Controllers, completing the feedback loop. Each server has a small piece of Server Manager code in it, which reports monitor information from the service application, and which relays control requests.

Figure 7 – The Monitor & Control System



Each service and each server reports its workload information to the Pool Controller in charge of the pool to which that server has been assigned. The Pool Controller aggregates this data and sends overall pool status to the Farm Controller, which keeps track of the abilities of the pools to provide the desired service levels to the customers using that pool. The Farm Controller also keeps track of any unassigned servers that are available. During normal operation, the displays associated with these control elements simply show the state, and allow administrators to adjust thresholds and other operational parameters. ***Figure 8*** shows what some of the display and control user interfaces (UI) look like.

Figure 8 – A Monitor/Control Example, Initial State (Low Load)



In our eUtility, the SLAs specify a threshold value for the latency of a message. The latency of a message is defined as the time it takes from when the message enters the system (at the eMail injector in ***Figure 1***) to when the message is sent to the user's device (the Notification Dispatcher sends the message to the SMS phone). Every pool in the farm has an SLA that it must meet.

The monitor and control system is lightweight; each element runs in its own Java Virtual Machine (JVM), and all may be run on the same computer, or may be distributed among several computers for the convenience of the service-provider. The UI elements may be run in the same way, and, in fact, multiple copies of the same UI element may be run, and each will be given the same data from the monitors. In the example above, the monitor and control elements are all being run on one computer, and the UI elements are being run on another (a "farm operations console"), from which the screen captures have been taken.

The Auto Farm Controller is constantly monitoring the pools and balancing the servers in the pools to maximize the difference in predicted latency from the latency threshold. (LatTh_n – LatPred_n). The latency is predicted from historical data and the last few readings of CPU utilization, network throughput, and message rates.

When the Auto Farm Controller determines that a server needs to be moved from one pool to another, it will send control messages to the appropriate Pool Controllers to reassign servers. These actions may also be triggered manually by an administrator to allow overrides for anticipated conditions or for servers to be removed for maintenance.

An Operational Example

In the example shown in ***Figure 8***, we have two pools and three servers. To make the example simple, we only have two customers, one, with a stricter SLA, assigned to Pool 1 (currently with one server) and the other assigned to Pool 2 (which has the other two servers). We have also simplified the workload for the purpose of the example, so that each work element completes very quickly and the monitor and response intervals are very small; with a normal workload and normal response settings, it may take longer to create, to detect, and to react to varying load conditions. In fact, the control system must be balanced in that regard, as too-fast response to a changing load can result in overreaction, and thrashing (moving a server back and forth between two pools, too often).

The first UI views show a normal load, with low CPU usage and good throughput. As the incoming message rate increases (***Figure 9***), the workload will increase on the servers until the server in Pool 1 may no be longer able to handle the requirement of its SLA. Here, the Farm Controller has determined that Pool 1 is too close to its SLA threshold, and that the workload in Pool 2 is such that a server may be removed from that pool without jeopardizing the SLA of the customer in that pool, so the Farm Control Manager will tell the two Pool Controllers to reassign Server 2 to Pool 1. The Pool Control Manager for Pool 2 will tell the server to stop work for that pool (***Figure 10***), static information needed to serve Pool 2's customer will be removed from that server, and information for Pool 1's customer will be put on it. When this process is finished, and Server 2 is no longer working in Pool 2 (***Figure 11***), the Pool Control Manager for Pool 1 will tell Server 2 to start work (***Figure 12***), work units will begin to be assigned to Server 2 from Pool 1 (***Figure 13***), and the Pool Sense Manager will register its listeners with the server. The server will have been switched, the workload soon balances out (***Figure 14***), and all servers are again operating within their capacities.

Figure 9 – Heavy Load, Pool 1 is Overloaded

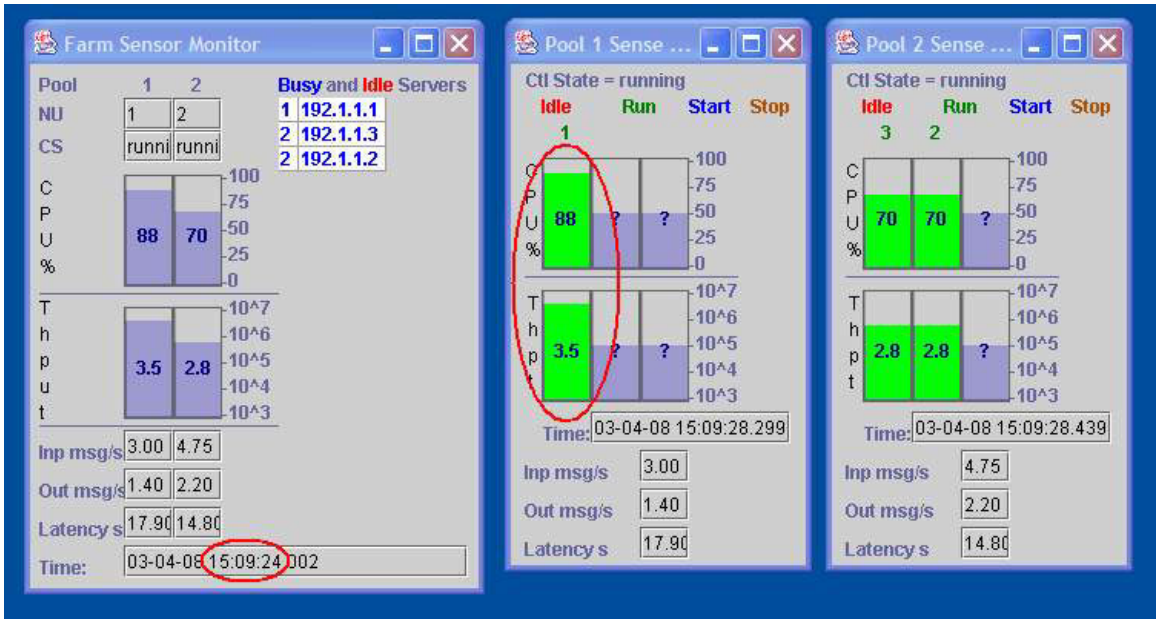


Figure 10 – Removing Server 2 from Pool 2

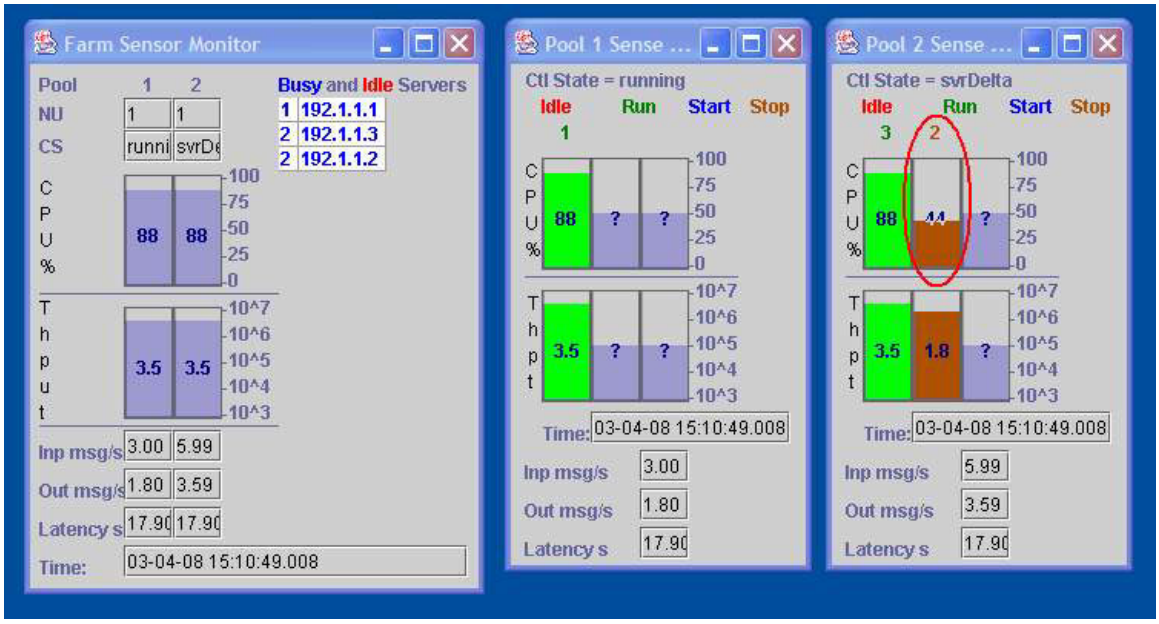


Figure 11 – Server 2 has Stopped

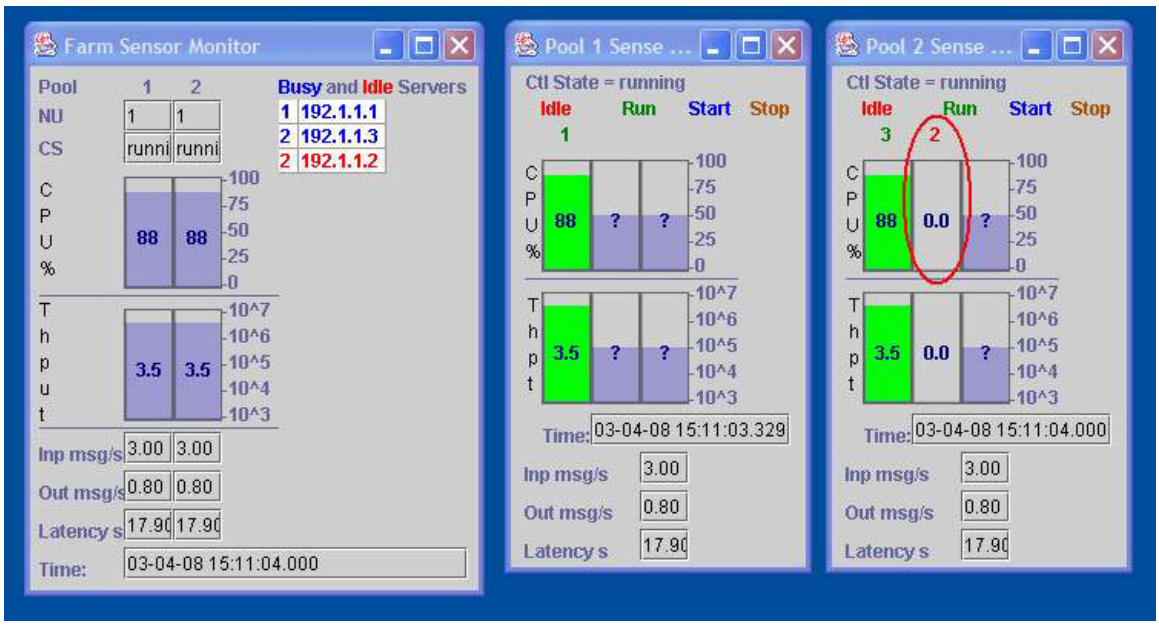


Figure 12 – Server 2 Reassigned to Pool 1

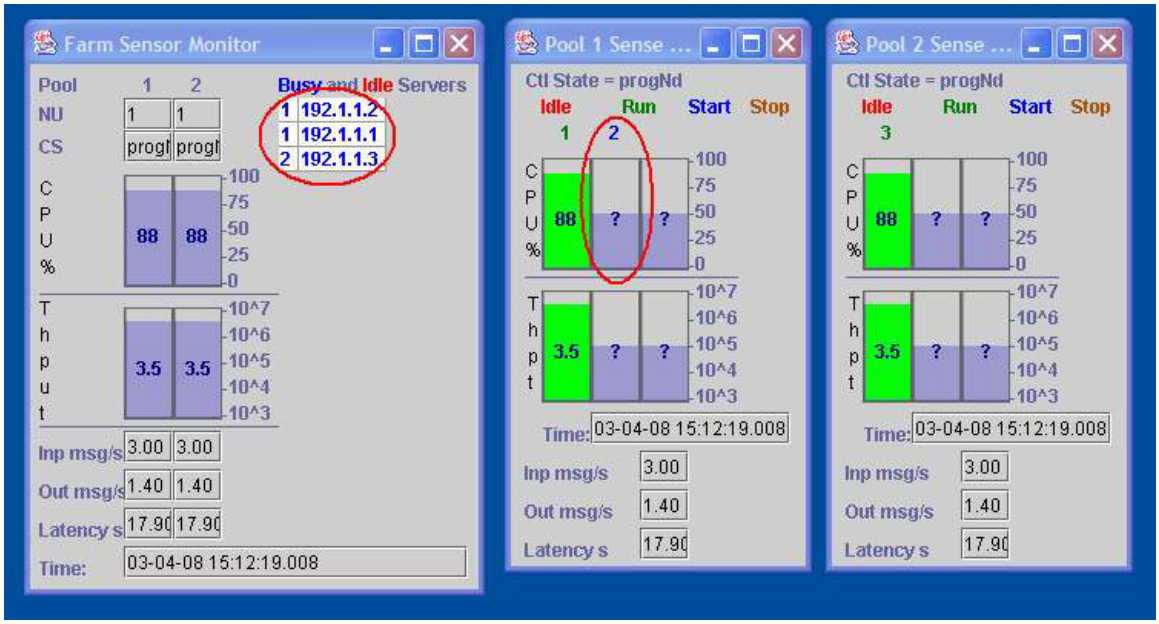


Figure 13 – Server 2 is Starting Work for Pool 1

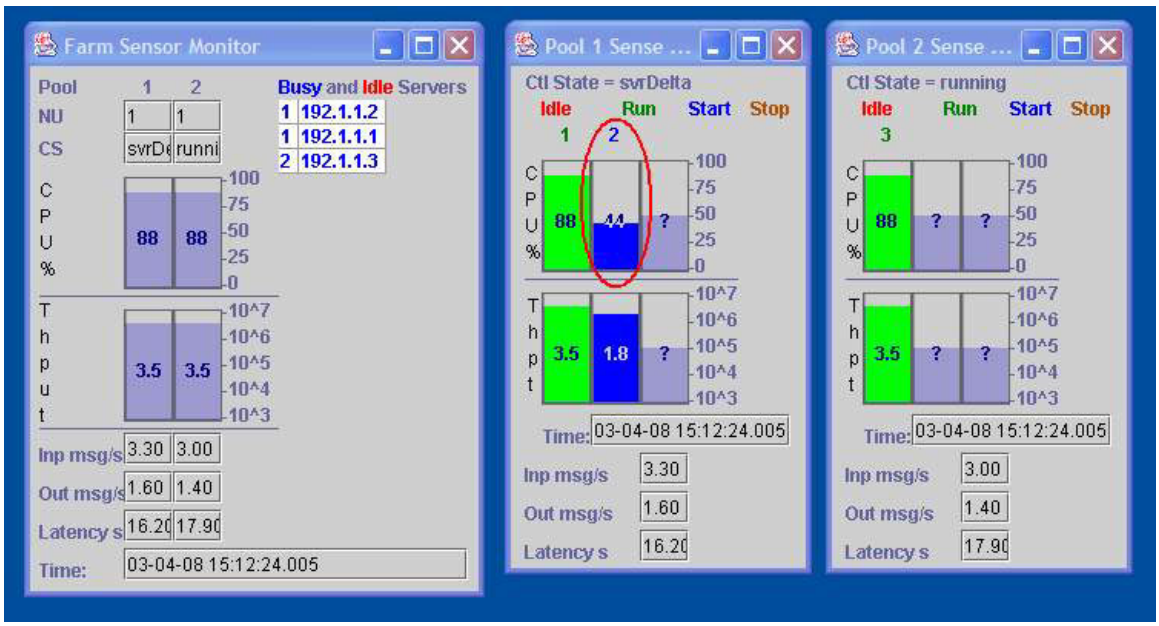
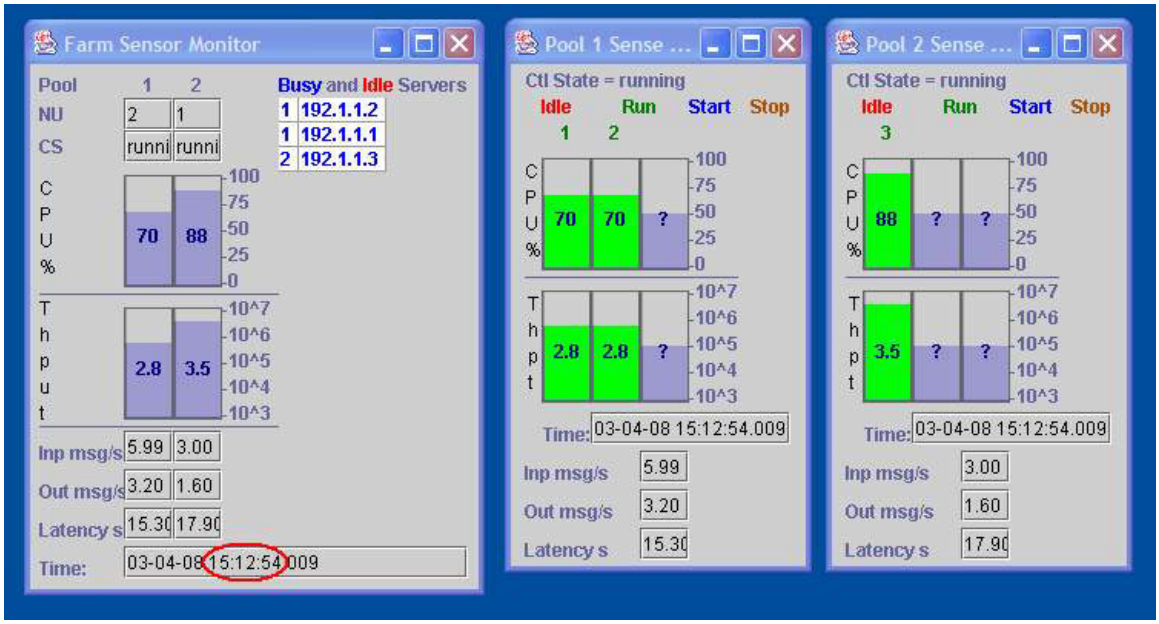


Figure 14 – Server 2 Takes Full Load



For manual changes to the configuration, each manual Controller has an administrative UI that allows such operations as removing a server from a pool, adding a server to a pool, and setting operations thresholds. When manual changes are made, the same operations happen as when the control system detects automatically that a change is needed.

Related Work

The ideas described in this paper cover the two areas of 1) automatic registration and provisioning of users in a computer utility environment and 2) dynamic resource control in a computer utility environment. In this section we position our work with similar work as reported in the literature. We also compare and contrast our project with the Océano computing utility project.

Automatic provisioning of users: We have not found any approach to user registration and provisioning, as reported in the literature, that is the same as that reported here. There are overall systems management frameworks or infrastructure in place that handle billing, SLA setup and monitoring, service sign-up, status reports, and similar ongoing interaction between utility service providers and enterprises that use the utility services. Our user provisioning mechanisms fit within such systems management frameworks, extending them. There is a lot of work on multi-service management systems.[3,8,9,10] The discussions on provisioning are generally limited to providing a web-based way for enterprise administrators to view the services offered, sign users up for services, and monitor the SLA metrics. None of them talk about how individual users within an enterprise are provisioned for services that require user-specific data.

Dynamic Resource Control: The resource control scheme used is based on feedback control theory and is similar to a set of emerging resource control mechanisms for multi-

service, shared resource systems. The ControlWare system[11] includes middleware for constructing feedback controls to meet quality-of-service (QoS) guarantees. Whereas the control mechanism we used was built from scratch to meet our needs, the goal of ControlWare is to build resource controls, based on feedback control theory, in a systematic manner. Adaptive control theory for QoS-aware computing has also been explored.[12]

Relation to Océano: The control system relates to work done on the Océano project,[13,14] in that they are solving similar problems from different directions. Where Océano focuses on monitoring dedicated machines, and switching machines from one service to another, our focus was on monitoring and adjusting a diverse set of applications that share processors, and this presented a different set of challenges.

First, since we don't attempt to install and remove software (though we do move configuration information and transient data required to handle a particular workload), all software must be installed and operational on all servers in the farm, so that any server may be moved quickly to another pool without manual intervention. Joining this with Océano could mitigate this situation and add flexibility to both systems.

Second, because Océano entirely replaces all data on a server when it moves that server, there are no inter-enterprise privacy issues involved. We did have to deal with those issues, as we shared computers between different customers, and such privacy questions must be further addressed in continuing work in this area.

Third, the measurements used to monitor different services are different. While there are common aspects of the system that we could and did monitor (CPU utilization in our example; others include memory usage, paging, total disk I/O, and those sorts of system-wide measurements), the best way to monitor any service is service-specific. We solved that by designing pluggable monitoring modules, implemented through a monitor API. The example here shows one service for simplicity, but in our work we used two services (the second was an information subscription service[15,16]), and using this API we could have separate monitors tracking the effective throughput of each service.

Conclusions

Our early research into building eUtilities showed that it is feasible to provide a common service to multiple enterprise customers, to isolate customers by putting the privacy-sensitive functions on the customer's side of the system, and to monitor service levels and system activity automatically, taking automated action to help ensure that agreements are met. Both components described here fit into IBM's Utility Management Infrastructure[17] in the subscriber management and enablement areas.

By allowing the customer to tie into its own existing employee directory, and to do user registration from there, we keep the portion of the system most likely to expose privacy and security concerns isolated and under the customer's own control. In addition, the customer has a convenient, efficient way to register users for the eUtility service, a way that allows traversal of the organizational structure and eliminates much of the error-

prone re-entry of information that's been common with registration for new services. Because we built the registration tool with plug-in modules, it's easy to add registration modules for several services, allowing a common interface for registration to a number of services that might be offered.

The monitor and control system we describe gives the eUtility administration an easy way to see what's happening with the services from a number of levels: overall in the server "farm", for each set of customers in server "pools", or for each individual service. The plug-ins here must, to be most effective, be tailored for the specific service provided ("throughput" measurements vary by service, for instance), but for any service we can measure CPU usage and other performance-related items (such as memory usage and paging) and provide notification or take action accordingly. This automated monitoring and control is important for providing a service that supports many customers, with a variety of needs and service contracts.

Tools similar to these, together with those that fill needs in the other areas of the Utility Management Infrastructure, will form the backbone of the developing "on demand" technology.

Acknowledgements

Much of the design of the monitoring and control system and the implementation of most of the components that were not specific to the notification service were due to our colleague Mike Spreitzer. Mike continues to work on performance management issues for large-scale services.[18]

Cited references

1. J. von Kanel, J. S. Givler, B. Leiba, W. Segmuller, "Internet Messaging Frameworks", IBM Systems Journal, Vol. 37, No. 1, 1998.

2. V. Bazinette, N. Cohen, M. Ebling, G. Hunt, A. Purakayastha, G. Sewart, L. Wong, D. Yeh, "INS: an Intelligent Notification System", IBM Research Report RC #22089, 2001.

3. "Service Creation: Mixing Up the Right Blend – Special Section", Telecommunications Magazine, Vol. 34, No. 1, January 2003.

4. A. Hiles, "E-Business Service Level Agreements", Rothstein Associates Inc., Brookfield, Conn. 2002.

5. Internet Engineering Task Force, "Internet Message Access Protocol – Version 4rev1", RFC 3501, Network Working Group, March 2003.

6. Internet Engineering Task Force, "Lightweight Directory Access Protocol", RFC 1777, Network Working Group, March 1995.

7. T. Howes, M. Smith, "LDAP Programming: Directory-Enabled Appplications with Lightweight Directory Access Protocol". McMillan Technical Publishing, Indianapolis, Indiana, 1997.

8. G. Rokosek, L. Lewis, "Dynamic Service Provisioning: A User-Centric Approach", 12th Int. Workshop on Distributed Systems: Operations and Management, October 2001.

9. D. Lewis, "A Review of Approaches to Developing Service Management Systems", Journal of Network and Systems Management, Vol. 8, No. 2, 2000.

10. G. Chen, Q. Kong, "Integrated Management Solution Architecture", IEEE/IFIP Network Management and Operations Symposium, Honolulu, Hawaii, July 2000.

11. R. Zhang, C. Lu, T. F. Abdelzaher, J. Stankovic, "ControlWare: A Middleware Architecture for Feedback Control of Software Performance", International Conference on Distributed Computing Systems, 2002.

12. Y. Lu, T. Abdelzaher, C. Lu, G. Tao, "An Adaptive Control Framework for QoS Guarantees and its Application to Differentiated Caching Services", 10th International Workshop on Quality of Service, 2002.

13. K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalanter, S. Krishnakumar, D. P. Pazel, J. Pershing, and B. Rochwerger, "Océano: SLA based management of a computing utility", Proceedings of 2001 International Symposium on Integrated Network Management, pp. 14-18, May 2001.

14. D. Pazel, T. Eilam, L. Fong, M. Kalantar, K. Appleby, G. Goldszmidt, "A dynamic resource allocation and planning system for a cluster computing utility", Proceedings of International Symposium on Cluster Computing and the Grid, Berlin, Germany, May 2002.

15. R. Strom, G. Banavar, T. Chandra, M. Kaplan, K. Miller, B. Mukherjee, D. Sturman, M. Ward, "Gryphon: An Information Flow Based Approach to Message Brokering", International Symposium on Software Reliability Engineering, 1998.

16. G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. Strom, D. Sturman, "An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems", International Conference on Distributed Computing Systems, 1999.

17. J. Nash, "The U Word at Big Blue", Newsweek, Vol. 140, No. 25, December 16, 2002.

18. R. Levy, J. Nagarajarao, G. Pacifici, M. Spreitzer, A. Tantawi, A. Youssef, "Performance Management for Cluster Based Web Services", International Symposium on Integrated Network Management, March 2003.