An End-User's Perspective on Application Management

Pankaj K Garg

HP Labs, 1501 Page Mill Road, Palo Alto, CA 94304, USA. garghpl.hp.com

We present an end-user's perspective on managing Internet applications. The end-user's perspective can: (1) guide the definition of metrics for which management data must be collected, (2) limit the amount of data collected for end-to-end management of applications, and (3) enable management by end-users themselves.

To explore concepts of management from an end-user's perspective, we have prototyped a tool, Amadeus, that enables management by end-users. Amadeus tracks end-user actions and correlates them with performance and business metrics. We describe the motivation and architecture for Amadeus. Two salient points about the architecture are: (1) data collection is triggered by end-user's interactions with applications (as opposed to a periodic polling scheme), and (2) application monitoring does not require source code access to applications. We illustrate our approach using the Lotus Notes email application.

Keywords: application monitoring, resource utilization, Lotus Notes, end-user monitoring, business metrics

1 Introduction

The Internet revolution of the past few years has enabled the rapid replacement of human and material processes by bits manipulated through computers and passed around through network wires and airwaves [DM98, Neg95]. Organizations have re-invented themselves and their supporting infrastructures for this digital revolution. A common software architecture for Internet applications enabling this digital revolution is illustrated in Figure 1. As depicted in the figure, a heterogeneous mix of software components of varying complexity execute on different platforms to enable a business transaction. While technologies for constructing such applications are improving rapidly (e.g., component software, Internet, and so forth), the management and operations of an application-in-execution is becoming increasingly complex and overwhelming. Numerous industrial analyses have documented the high costs of managing and operating computer based networks and services. Gartner estimates that application disruptions costs business $200 billion a year (`www.firstsense.com`). Our research goal is to develop tools and methodologies to mitigate the cost of managing and operating computer-based services.

One approach for applications management is to extend the traditional approaches for network management towards application management. The traditional approach for network management is illustrated in Figure 2. As the figure depicts, a common assumption in network management is that a network manager monitors and administers a computer network. To achieve scalability in this approach, therefore, the network is subdivided into domains that can be easily managed by a single operator. This limits the size of a typical domain to about a few thousand network elements. Even with a few thousand elements, however, an operator is often overwhelmed with the icons on a topology map, or the number of events that he has to watch over. Technologies such as 3-D visualization (`www.cai.com/products/unicent/tng_brochure.pdf`) and event correlation (`www.openview.hp.com`), although useful, are not able to adequately address these problems.

One problem with extending traditional network management approaches to application management is that of **scale**. With the commercialization of the Internet, organizational boundaries have become somewhat fuzzy. Application end-users can be located anywhere in the world and can come in numbers in the millions

O. Festor and A. Pras (Eds.) : 12th International Worshop on Distributed Systems: Operations and Management DSOM'2001 Nancy France, October 15-17, 2001.

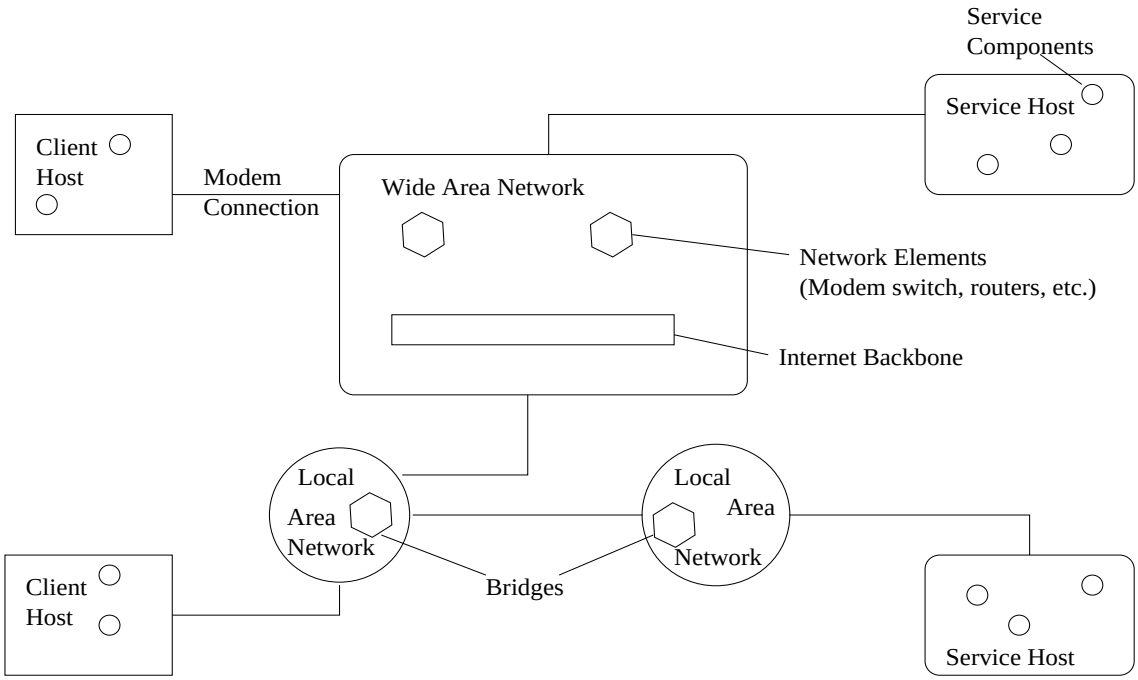

Fig. 1: A typical Internet application with several hardware and software components.

and eventually even billions. With number of users such as these, it becomes impossible for a centralized, system administrator oriented management approach to work. That is why current solutions to application management mostly adopt a server-side management approach. They manage the server side of the application. Even the ones that do perform some end-user monitoring have to limit their monitoring to a few user samples.

Often, the server-side management approach delegates management of user-perceived metrics to customer-support centers. This only shifts the management burden to a logically centralized customer-support who in-turn get overwhelmed with the number of complaints and calls from end-users.

Privacy of information collection is another issue with server-side management of applications. End-user's may not want information about their application use to be broadcast to administrators. This problem gets more serious as the metrics collected become closer to the businesses supported by the applications.

Finally, **information loss** for a particular end-user's transaction with server-side administration is another problem with server-side administration. For example, when statistically averaging response times over a class of users, the administrators may miss the particularly bad response time a given user is experiencing. Moreover, diagnosing the cause of bad response times is even more difficult with statistical measures, because we lose the correlations between sub-transactions.

A novel approach to solving these problems is to turn the management solution around: instead of using a system administrator's perspective on managing applications, view the application management problem from an end-user's perspective. As shown in Figure 3, this approach is a much more scalable solution than that of Figure 2. Here, *instead of one operator having to manage the relationships among software components of millions of users, each of the millions of users manages the relationships and services offered through the few components that his transaction requires.* An analogy of a working solution is the Internet package tracking mechanism available to customers of Federal Express (`www.fedex.com/us/tracking`).

As with Federal Express, however, we must ensure that information and control is presented to the end-users in their terms and made convenient to use. Hence, in order for this approach to be successful, we must address a few challenges: (1) what are the relevant metrics to collect from an end-user's perspective?

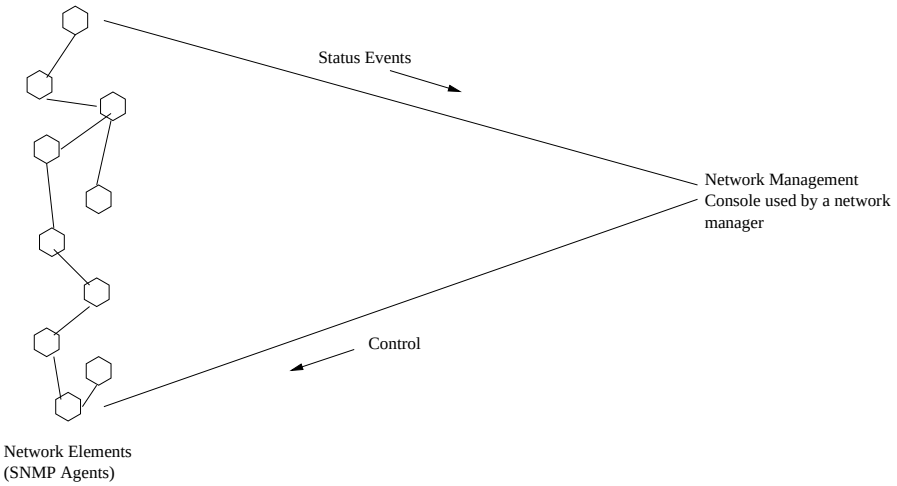

Fig. 2: A typical network management scenario: multiple network elements and their topology (connections) are managed via a central management console.

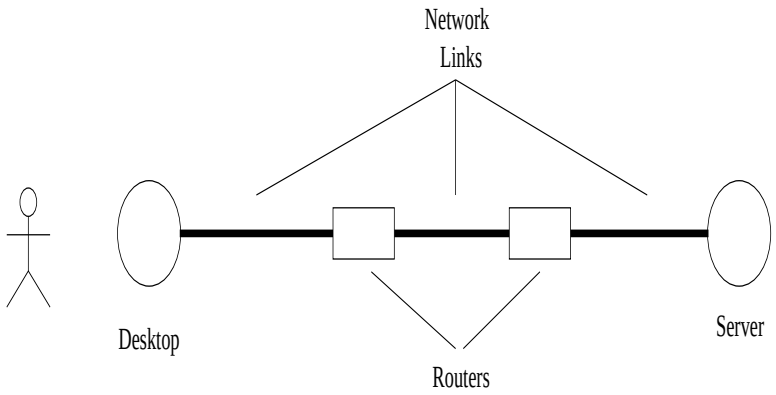

Fig. 3: A typical path for a distributed application from an end-user's point-of-view.

and (2) how can we present information to end-users to enable them to carry out more management tasks. We provide a solution for these problems in the rest of this paper. We describe the concept of end-user and application "interactions," and management data categories in Section 2. We describe a prototype system, called Amadeus, that explores management capabilities for end-users. Section 3 gives an overview of Amadeus. Section 4 describes the architecture of Amadeus. It describes the user action tracker module in detail. Section 5 describes some related work. Finally, in section 6 we present some conclusions and directions for future work.

2 End-User and Application Interactions

End-users **interact** with their applications. Opening an "Inbox" folder, or composing email messages are examples of interactions between a user and an email application.

An interaction starts when the user presses a keyboard key or selects a menu item from an application menu. The start of an interaction results in computing and possible I/O activity at the user's desktop. Subsequently, a request from one of the desktop components may be routed via a digital network to a server component executing on a server machine. The server component, in turn, executes some instructions on its machine and possible accesses devices such as disk and CD ROM. This pattern of a component performing

some work and asking a server component to do some work can go on recursively for several tiers.

Within this flow of execution, the user may again get involved and provide input for the system to continue processing the interaction. Finally, at some point the interaction ends with either new data being displayed on the user's application window or the user selecting a new application command. Figure 4 shows an abstract representation of the control flow of an interaction.

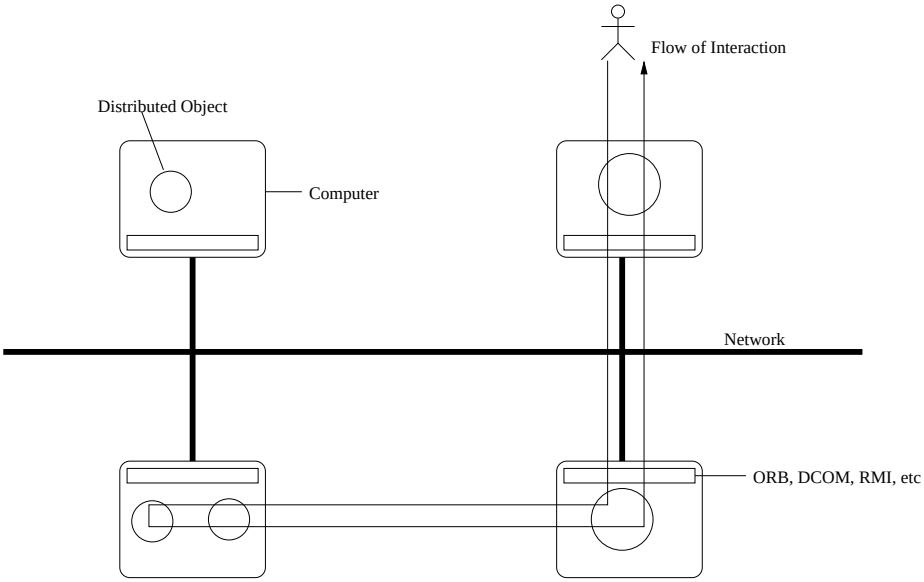

Fig. 4: Flow of control and data, from an end-user viewpoint, of user/application interactions.

2.1 Management Data

From a management viewpoint, we want to collect data that will enable end-users to monitor their interactions with applications and take corrective actions in case of unsatisfactory data values. Broadly speaking, we have data that can answer two categories of questions:

- Business (Productivity) Data: What end-user resources are being spent on a class of interactions?
- System Data: How and what system resources are being used for an interaction?

2.1.1 Business Data

The business data will depend on the user processes that are enabled by the application interactions. For example, email applications enable communication and coordination processes in organizations. The steps in these processes are: read message, open folder, compose message, send message, detach attachment, and so forth. Each of these steps requires the user to spend time with the application. The duration of these interactions, therefore, becomes a basic metric for management. Similarly, the event of an interaction occurrence is a basic metric. From these basic metrics, we can derive other useful business metrics. For example, what percent of interaction time was spent in composing email messages? How many messages did a user read in a given day? In how many instances did a user reply to messages?

2.1.2 System Data

System data consists of two parts: Configuration and Performance. We discuss each of these in turn.

- **Configuration:** Is the computing infrastructure in place for the end-user to execute an interaction through the application? Why did the last interaction not execute to completion? Where did it break?

If some component is not available for an interaction, what is the most efficient way for the end-user to make it available? The main issues with application configurations are:

1. Platform Coverage: Which applications in what versions are installed on the end-user's machine, e.g., desktop?
2. Application Components: What are the components (executable and data files) that any particular application requires to execute properly? Where are those components on the desktop? Are the component versions appropriate for the application to execute? Are the data files configured properly for the application to execute?
3. Application Servers: What server components does the application require? Where are those servers located? Is the desktop configured properly to call upon the services of the server?

Note that some of these questions may not be in the form posted by an application user. The application user may just want to get his application to work. The answers to these questions, however, are necessary to enable an end-user to check the correctness of their desktop and server configurations. If the need arises, the answer to these questions can substantially reduce the time-to-diagnose for configuration related problems.

- **Performance:** How much time did the last interaction take to execute? What is the average time that interactions of this nature typically take? Why was the performance of the last interaction so slow? How can the end-user get the interactions to perform faster?

Figure 5 shows a typical resource consumption graph (RCG) for a distributed, client-server application. We expect the end-user to collaborate with the systems administrator to effectively manage the resources along an RCG.

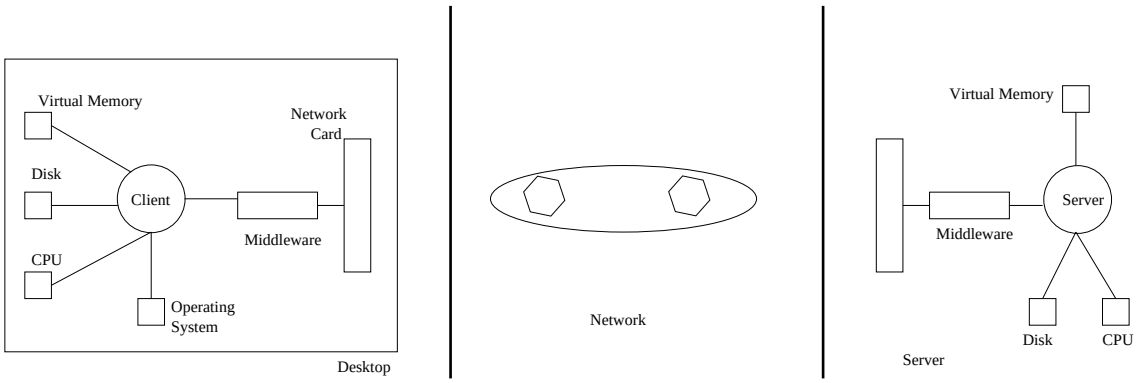

Fig. 5: A Typical Resource Consumption Graph for a client-server application. To keep it simple, we study a distributed application with one server for many clients. A client accesses shared resources like virtual memory, disk, window system, middleware (RPC runtime system) and the network card. A similar resource sharing picture emerges for the server. Any of these shared resources could be a transaction bottleneck.

As the figure shows, a client application process requests the services of either the CPU, the disk, the operating system, or the network. Each of these resources are being shared among the client process and other processes running on the same host. A similar situation occurs for the server and the network. One important factor in the performance of an interaction is the utilization of the various shared resources. So, we monitor the utilization of the various shared resources. In some cases, we can also compare the utilizations of shared resources by a given interaction with the utilization by other activities. For example, in the case of the Notes application on a user's desktop we can compare the CPU utilization by the Notes process to the CPU utilization by other processes. This kind of comparison is more difficult, or even impossible, for other server-side shared resources like the network or the server. Eventually, we will have to develop standards for servers to enable collection of such metrics.

3 Amadeus: A Tool for Management by End-User

To develop the concepts for an end-user's perspective on application management, we have defined a tool, Amadeus, that enables management by end-users. Figure 6 shows the main screen for Amadeus.

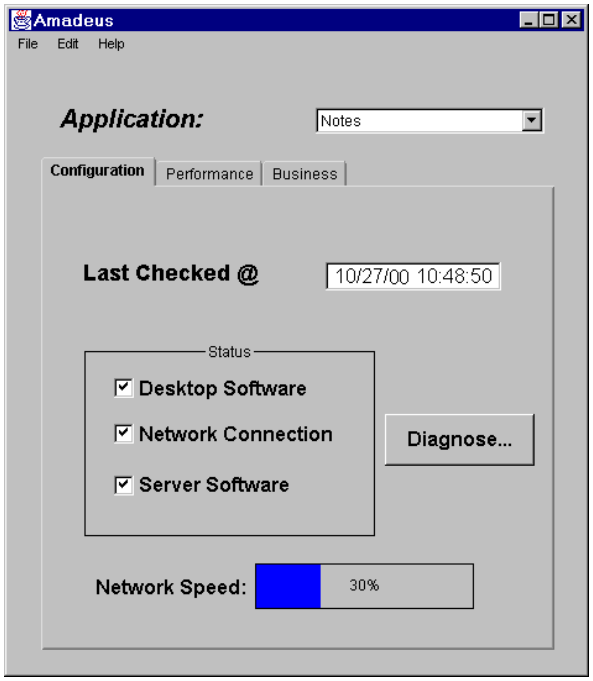

Fig. 6: Top-level screen image for Amadeus. The user selects an application to be managed, from the top right drop-down list. Subsequently, the user can monitor information about the application's configuration, performance, or business use.

As shown in the figure, a user selects one of three tabs for management: (1) Configuration, (2) Performance, or (3) Business. The user can choose to manage one of the many applications running on his desktop, as shown by the drop-down list of applications on the top right. When the user selects the Configuration tab, Amadeus performs various configuration checks to ensure that all components required to run the application are available to the user. Amadeus checks the user's desktop, the network, and the server for configuration related information. On a Windows desktop, Amadeus checks for the appropriate DLL's and their version numbers. For the network, Amadeus checks to ensure that the desktop is connected to the server via a network connection, and reports on the average network bandwidth. Finally, for the server, Amadeus ensures that all the processes required for a successful application connection are running. If any of these checks fails, Amadeus will not mark its checkbox. At this point, the user can push the "Diagnose" button to get more information. For example, Amadeus will inform the user of any missing DLL's, or where the network is broken, or if the server is broken. If Amadeus cannot perform the diagnosis or repair itself, it may assist the user in submitting a service request to an appropriate system administrator. Such configuration checks and diagnosis will be based on administrator specified configurations.

Figure 7 shows a screen image for the "Performance" tab. Here we see either a real-time graph for current performance, or a graph of performance logged over some time period. Several salient aspects of this performance display are:

1. On the same graph we see interaction metrics and system metrics. For example, we see the response time for a given interaction (open message), with %CPU utilization, memory utilization, and so forth. The metrics that are shown here can be configured by the user to include other metrics such as network bandwidth, server utilization, and so forth.

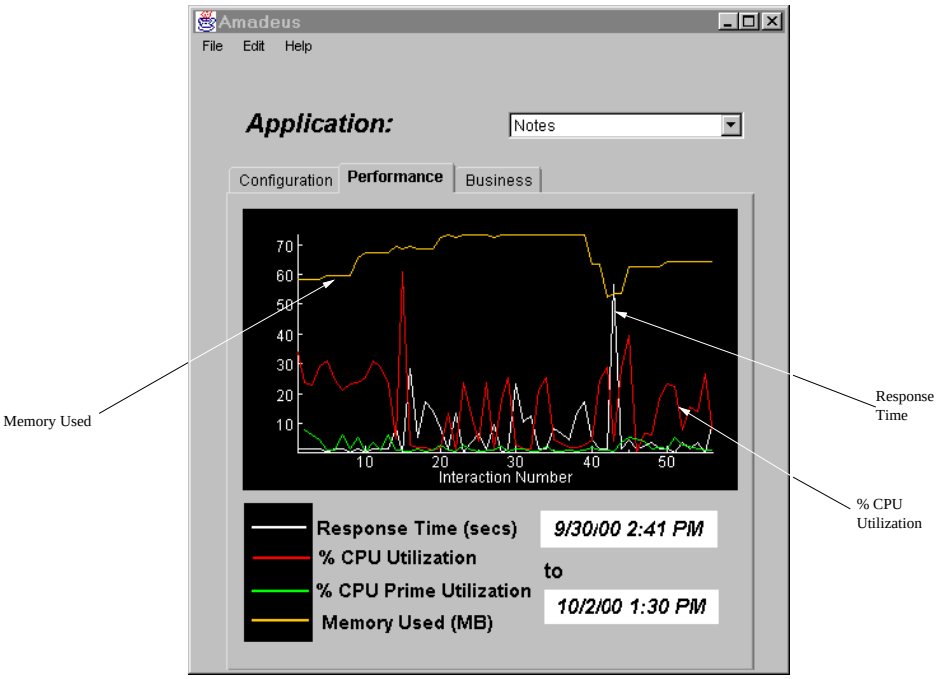

Fig. 7: An "interaction series" view of the performance data for the "Open Message" interaction for a run of the Notes application over two days.

2. The graph shows data per interaction rather than at some periodic intervals. Hence, we can show data for about two days in a small graph, with reasonable resolution. Moreover, the data is mostly about the given application and the end-user's interaction with it. There is no extra data for the durations when the interactions were not happening. In comparison, Figure 8 shows a time-series plot for the same data. Comparing the two graphs, we suggest that the interaction-series display can provide more correlation information for end-user interactions.

3. Notice the spike in response time in Figure 7 comes with a spike in the CPU utilization and a drop in the memory used. This suggests that the application is performing some kind of garbage collection, and hence the user should not be concerned about the drop in performance.

Finally, Figure 9 shows a screen image for the "Business" tab. Here we report on some productivity metrics for the user's interactions with the application. For example, over a two day period this user sent seven messages while he read fifty-six messages. Maybe this indicates a communication problem for the user? In addition, the user did only two folder switches in the entire period. This may indicate the lack of good use of folders? It seems that these kinds of business metrics can be useful for:

1. Analyzing the user's interactions with the application to make the user more effectively use the application for his ultimate process or business goals.

2. Enable the user to configure the application to better suit his needs. If the user finds that he really does not use the folders capability in an email application, then why should he have to pay for it? For example, the end-user's machine is probably wasting main memory and disk space for this additional feature.

3. Application designers can use this information for their future releases of the application, or for "target marketing."

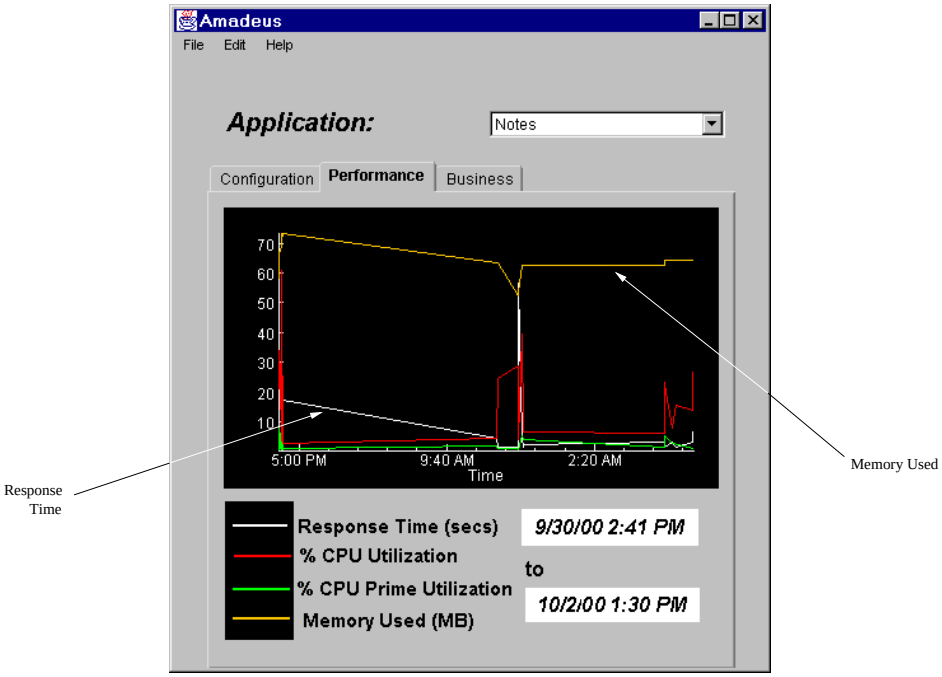

Fig. 8: Performance data shown in a time-series instead of an "interaction series."

4 Amadeus Architecture

Figure 10 depicts an architecture for enabling applications management by end-user. As the figure illustrates, the main components of the architecture are:

- Application and Interaction Description
- Configuration manager
- Performance monitor
- User Action Tracker
- Data Agents

The Data Agents provide configuration and performance data from the application's client, network and server. We've built some specific network data agents using ICMP (Internet Control Message Protocol) and SNMP (Simple Network Message Protocol). For most of the client and server information we rely on the Perfmon data available through the Windows NT registry [Bla93]. In the future, we expect to integrate this with additional API's like Measureware (`www.hp.com/openview/rpm/perf.html`). Perfmon counters keep raw data and the "reader" of the data value is required to interpret the data values. For example, Perfmon has a cumulative counter for the CPU time used by a process. To derive the CPU utilization in a given time interval $[t_2, t_1]$ we can poll this counter at times t_1 and t_2 obtaining values C_1 and C_2, respectively. The utilization during the interval then is $(C_2 - C_1)/(t_2 - t_1)$. This feature of Perfmon enables us to calculate the utilizations for arbitrary and overlapping time intervals. This works for the client side, but doesn't work for the server side as there might be a delay from when a user action starts and when a server can be notified. We are working towards a solution for this problem.

The **Performance Manager** collates the information from the data agents to present through a graphical user interface.

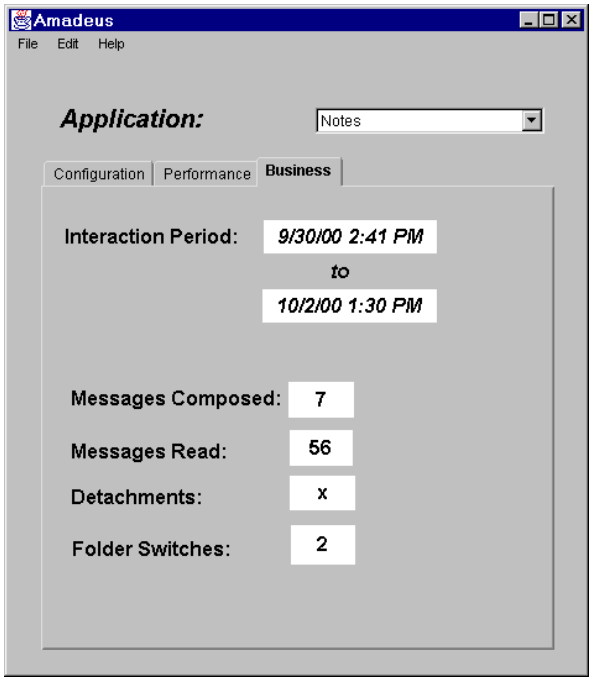

Fig. 9: Screen showing some productivity metrics for a run of the Notes application over a period of two days.

The **Configuration Manager** reads a description of the application and checks the desktop file system and registry for application information. It also collects information about the application available to the user.

The **User Action Tracker** is the most novel aspect of Amadeus. Hence, we describe it in detail in the following subsection.

4.1 User Action Tracker

Interactions are characterized by two events: the start event and the end event. A start event is the user-initiated event that causes the interaction to happen. The end event signifies the end of an interaction for some management function. For example, when a user pulls down a menu on an email application to start composing a message, the start event is the selection of the command item that issues a command to the application to allow the user to start composing a message. When the user selects a command button to send the recently composed message, it signifies that the interaction "compose message" has ended. The duration of an interaction is defined as the elapsed time between the start and end events of the interaction. For interactions that do not require user interaction beyond the start of the interaction, the duration is the response time of the interaction.

Often, users are not cognizant of the durations of their interactions with applications, e.g., how much time does it take my email application to display the contents of a selected message? Amadeus provides a mechanism for the user to track the duration of key interactions. The main challenge in doing this is to track interaction durations for applications without having access to the application's source code. Otherwise, we may develop schemes that are not generally applicable. We cannot always rely on instrumenting the source code to collect interaction durations.

The general scheme for obtaining an interaction duration is to monitor its start and end events. Each time an interaction start event is observed, we record a timestamp for the event. Each time an interaction end event is observed, we record a timestamp for the event. The difference between the two end event and start event timestamps gives us the interaction duration. Hence, interaction durations can be determined if we can monitor the start and end events of durations.

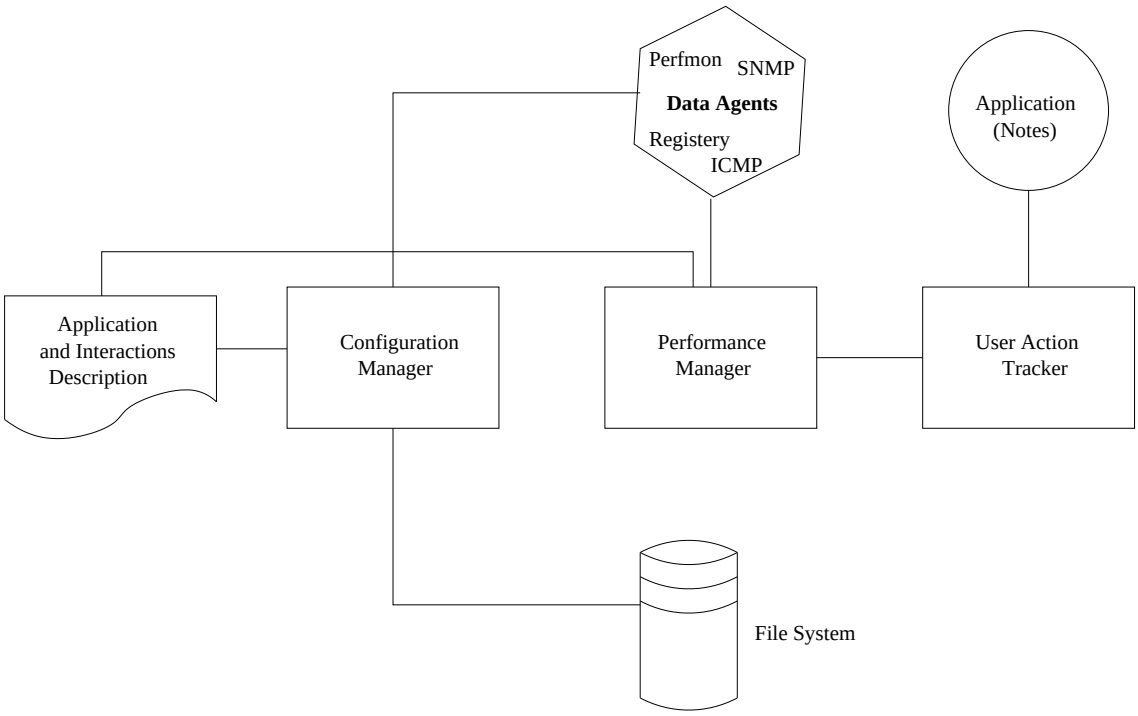

Fig. 10: Amadeus Architecture: These are the main Amadeus software components that reside on either the end-user's desktop or on the server.

There are two general ways of identifying interaction events: (1) binary code instrumentation [XMN99](also see `www.appliant.com`), and (2) windows event monitoring [Ric97]. Code instrumentation observes the binary code execution of the application, and based on repeated patterns of executed code identifies signatures for particular events. Windows event monitoring observes the behavior of messages traveling between the application and the graphical user interface, and based on repeated observation of the message flow, identifies patterns of window messages that act as signatures of particular events. We use the message observation method for this implementation of Amadeus. In the future we will explore binary code instrumentation as well.

In the Microsoft Windows environment, we can look at messages flowing between the application and Windows and obtain interaction durations. Figure 11 shows a typical configuration of how messaging works in Windows. Whenever the user interacts with an application, or a application interacts with the user, certain messages flow between the application and the Windows subsystem. For example, when the user selects a command by using the mouse left-button click, a message of type WM_LBUTTONDOWN flows between the Windows subsystem and the application. Similarly, when an application has finished drawing new contents in one of the application's window, a message of type WM_PAINT flows between the application and the windows subsystem.

To identify the messages corresponding to the start and stop events of an interaction, we run the application and at the same time observe its message flows using the Microsoft Spy++ utility. After a series of runs, the start and end messages become obvious for each interaction. For example, we observe the following interactions for Lotus Notes email application:

- Open mail
- Open mail folder
- Detach a mail attachment
- Compose mail

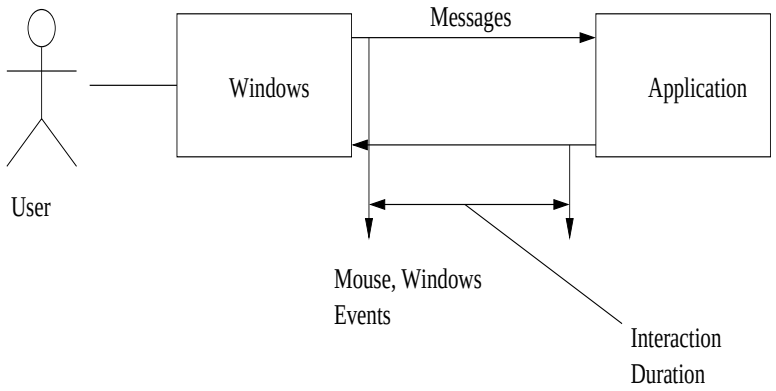

Fig. 11: When a user interactions with a Windows application, important events about the interaction can be monitored by monitoring the messages flowing between the Windows subsystem and the application.

The open mail and folder interactions start with messages for the same window class: NotesSubProg. The open mail interaction begins with a WM_LBUTTONDOWN (left button click) and ends with a WM_PAINT (done window painting) message. The difference between the two is that the open mail messages start and end on a window with a title whereas the open folder messages start and end on a window with no titles. The compose mail interaction starts when the user clicks on one of the Reply, Reply With History, Forward, or New Memo buttons. The compose mail interaction ends when the user clicks on the "Send" button.

Identifying the start and stop messages for Windows applications is non-trivial task requiring may be a couple of weeks of observations. Fortunately, this has to be done only once per application and then can be used by all the users of the application.

5 Related Work

Frølund et. al. [FJP98] define SoLOMon, a generic approach for a scalable, extensible distributed monitoring system. Their goals are to provide a programmable infrastructure to enable end-to-end performance management of distributed applications. Some of the Amadeus features can be implemented using SoLOMon concepts, e.g., description of server-side performance metrics. In several ways, SoLOMon needs to be extended to provide the full capabilities of Amadeus, e.g., the distinction among performance, business, and configuration data. A key difference between the two approaches is that SoLOMon advocates scalability by data traffic reduction by using mathematical abstractions over time-we advocate data reduction by limiting the data transmitted to the duration in which an interaction transpired.

Vital Signs (`www.ins.com/knowledge/whitepapers/characterizing.asp`) is a newly formed company that provides a tool, Net.medic, with goals very similar to ours. Similarly, FirstSense, Inc., is a start-up focusing on distributed applications management (`www.firstsense.com`). Vital Signs provide an end-user perspective on performance metrics for Internet- based client-server applications. Hence, Vital Signs is more focused on providing solutions for Internet-based and browser-based applications. Its not clear if the same techniques can be readily adopted for enterprise applications. FirstSense, Inc., however, does provide an end-user's perspective on enterprise applications.

Jacobson's (`ftp://ftp.ee.lbl.gov/pathchar`) and Praxon's [Pax97] work for network monitoring is similar in spirit to our work for application monitoring. In fact, some of the algorithms that are used in their work for network monitoring can be used for monitoring the network aspects of applications monitoring. Similarly, the cross-industry consortium is developing some standard metrics for monitoring networks from customer's point-of-view [Cro98].

6 Conclusions and Future Work

We have described an end-user's perspective on managing distributed enterprise applications. We illustrated the concepts by developing a prototype tool Amadeus and illustrated its application to the Lotus Notes email application. Our main conclusions from this work are:

- A monitoring and display system based on end-user actions can significantly reduce the amount of data collection and clutter in a display graph.
- Monitoring end-user actions can lead to new interaction metrics that can be related to user's and organization's productivity and to future application designs and deployments.
- An architecture based on management by end-users is much more scalable with respect to number of users than an architecture based on central administrator management.

There are several directions for future work in this area:

- Integration of an help-desk or service-request management system with Amadeus.
- Separating an application's logic from the management system, so that the same infrastructure can support a new application easily.
- Integrating server-side metrics more tightly with the client-side metrics. Develop standard channels to expose server metrics to clients.

We will be pursuing work in this direction.

Acknowledgements

I thank Svend Frølund for his discussions and help with developing some of these ideas.

References

[Bla93] Russ Blake. *Optimizing Windows NT*. Microsoft Press, 1993.

[Cro98] Cross-Industry Working Team. Customer View of Internet Service Performance: Measurement Methodology and Metrics. See http://www.xiwt.org/documents/documents.html, September 1998.

[DM98] L. Downes and C. Mui. *Unleashing the Killer App: Digital Strategies for Market Dominance*. Harvard Business School Press, Boston, MA, 1998.

[FJP98] S. Frølund, M. Jain, and J. Pruyne. Solomon: A distributed service level objective monitoring framework. Technical Report HPL-98-117, HP Labs, 1501 Page Mill Road, Palo Alto, Ca 94304, 1998.

[Neg95] N. Negroponte. *Being Digital*. Alfred A. Knopf, New York, NY, 1995.

[Pax97] Vern Paxson. *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, University of California, Berkeley, April 1997.

[Ric97] Jeffrey Richter. *Advanced Windows*. Microsoft Press, Redmon, WA, 1997.

[XMN99] Zhichen Xu, Barton P. Miller, and Oscar Naim. Dynamic instrumentation of threaded applications. In 7^{th} *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programmin*, Atlanta, GA, May 1999.