In: *New Developments in Distributed Applications and Interoperable Systems: 3rd IFIP International Working Conference (DAIS 2001), Cracow, Poland*
Kluwer Academic Publishers, September 2001

SERVICE ORIENTED APPLICATION MANAGEMENT — DO CURRENT TECHNIQUES MEET THE REQUIREMENTS?

Rainer Hauck, Igor Radisic

Munich Network Management (MNM) Team, University of Munich, Dept. of CS
Oettingenstr. 67, D-80538 Munich, Germany. Phone: +49-89-2178-2155|2167, Fax: -2262
{hauck|radisic}@informatik.uni-muenchen.de

Abstract Besides the conclusion of agreements about quality of service (QoS), one of the major prerequisites for a global service market are means to monitor the fulfillment of those agreements. From a user's perspective, the time needed to complete a service transaction represents one of the most critical QoS parameters. As most electronic services are usually based on distributed applications, obviously the same techniques can be used to measure the performance of electronic services as well as the underlying applications. In recent years several techniques evolved to monitor the application performance. However, the new aspect of service orientation adds relevant new requirements that are to be posed on such solutions. This paper evaluates the various techniques from a service oriented point of view and presents open research questions.

Keywords: Distributed Applications, Application Performance Monitoring, Service Performance, Electronic Service, Quality of Service

1. INTRODUCTION

As the analysis of agreements between providers of so called *electronic services* (e–services) and their customers shows, one of the most important prerequisites for a successful business relationship is the fulfillment of the agreed quality of service. Usually several problems arise from not fulfilling the agreement that lead to consequences ranging from penalties to be paid by the provider to breaking up the agreement. The discipline of *service management* provides and develops tools, techniques and methodologies to support the provider in fulfilling his agreements.

As current e–services are usually based on distributed applications (*application services*), it is actually possible to use the same techniques to manage both the e–services and the underlying applications. As a consequence there are several new requirements emerging from service orientation in application management: the main focus changes from application deployment and configuration to monitoring and controlling proper fulfillment of the agreed QoS. From a customer's point of view the most interesting QoS parameter in this context is the time needed to successfully complete a transaction. Taking too much time resolves into user's discontent which is to avoid. As providers are strongly interested in fulfilling the agreed QoS parameters, the ability to react in time is most relevant to the provider in case application performance degradation is noticed and thus the violation of an SLA is possible. On the other hand customers want to verify the fulfillment of their SLAs. Thus, for both sides means to monitor the duration of transactions are necessary.

In recent years several techniques evolved to monitor the performance of distributed applications. This paper presents a classification and a criteria–based rating of existing approaches in the area of application performance monitoring from a service oriented point of view. Furthermore, the analysis is used to identify and structure open research questions.

The requirements to be posed on the different approaches have been derived from a generic service model [8]. As already mentioned, one of the most important requirements is to monitor **service-oriented parameters** (e.g., transaction duration, service availablity). Of similar importance is the monitoring of **actual user experience** (as opposed to drawing samples) and the availability of **in-depth information** to allow providers to easily identify root causes of problems. Further requirements are **minimal effort** for all participants, **minimal performance impact** on the application service to be monitored, the possibility to do **real-time monitoring** as well as **general applicability** of the solution (concerning e.g., operating systems, programming languages).

2. CLASSIFICATION OF APPROACHES

In recent years a number of approaches to application performance monitoring have evolved. To alleviate evaluation of specific techniques, the following section introduces a classification gained from a survey of currently existing solutions by applying the requirements mentioned above. Eventually, the most prominent representatives of the different classes – both current products and existing standards – are briefly introduced. Figure 1 shows a simple model of a distributed application as well as its underlying infrastructure. The measurement of performance parameters can either take place at the application itself or in the infrastructure (both at the network and the systems). When measuring the application itself, approaches that solely focus on the client–side of the application can be distinguished from those taking into account the application as a whole (client– *and* server–side). This leads to the following four classes of techniques: *Monitoring of network traffic, system–level monitoring, client–side application monitoring* and *application–wide monitoring.* The following paragraphs explain these four techniques along with a number of subordinate techniques in greater detail.

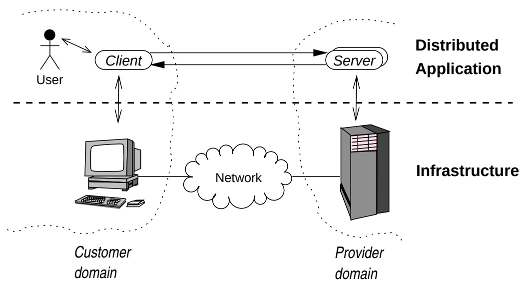

Figure 1 Distributed application scenario

2.1. MONITORING OF NETWORK TRAFFIC

A wide spread technique to identify QoS problems of applications in networks is scanning the network traffic in order to detect transaction–like behavior. Looking at IP–based networks, traffic occurring successively between the same pair of source and destination address is called a *flow* that, in combination with the application's port number, helps to find out start and end point of a client request and its response. In case of high–speed networks an evaluation of the measured data mostly cannot be done on–the–fly and therefore flows are recorded in real–time but analyzed at a later date. If network nodes (e.g., routers) do not support recording of flows, especially suited devices (so called *probes*) are installed in the network that provide the needed functionality. Advantages of this method are that no source code access is needed and every application that communicates over the network can be monitored.

Nevertheless, there are several disadvantages. As usually only transactions of standard applications can be detected automatically the system administrator must have in–depth knowledge about protocols of custom–designed applications to configure probes in that way that they distill individual transactions. Even worse, many protocols are used for different purposes than originally designed (e.g., SOAP uses *HTTP* for its RPC mechanism), which further increases complexity. In case of encrypted communication monitoring of network traffic is not suitable at all. As mentioned above, due to massive data volumes in high–speed networks the data analysis cannot be done in real–time and therefore cannot be used to react in time when a problem occurs. Additionally, it is impossible to determine the reason of a QoS degradation as long as it is not a network failure. Furthermore, time stamps are not taken from a user's point of view but at the network meter point.

Overall, solely monitoring network traffic is not suited well for application service performance monitoring as this technique was originally developed and used to detect network failures, for capacity planning and for reporting. There are several working groups within the IETF that develop standards in the field of network based (application) performance monitoring: e.g., *IP Performance Metrics (IPPM)* [13] and *Realtime Traffic Flow Measurement (RTFM)* [3]. Several companies offer probes and analyzing software using this technique: e.g., *CompuWare's EcoSCOPE* [6], and *Apptitude's MeterFlow* [2].

2.2. SYSTEM-LEVEL MONITORING

A second approach to application performance management is to measure system–level parameters like CPU usage, memory utilization, number of open files or run state of a process or a thread. By mapping processes and threads to applications, status information about the application is gained. The main advantage of this approach is the great experience gathered over the last years which allows easy collection of the data. It is relatively easy to read this kind of information from the system and provide it to management systems via well–defined interfaces.

However, there are major drawbacks of this technique as well: The basic problem of this solution is that it cannot provide the QoS parameters agreed with customers. Information about the state of the underlying systems is of minor importance to technically not versed customers. Mapping of system–level information to user–oriented parameters is often impossible. Even if a process is running perfectly from a systems view, the transactions a user is interested in might still be failing. Therefore, the monitoring of system–level parameters can be invaluable for a provider to monitor overall system performance but cannot be used for measuring actual application performance and verifying SLAs.

The IETF makes heavy use of this approach by providing a number of MIBs (e.g., SysAppl MIB [12]) concerning the area of application management. A lot of management tools, like HP Perfview [10] also follow this approach.

2.3. CLIENT–SIDE APPLICATION MONITORING

Monitoring solely from client's side is the third class of techniques we discovered. In contrast to the methods mentioned so far it is possible to measure the actual time an application needs to complete a transaction, i.e. it is metered from a user's perspective. Nevertheless, this class of techniques still suffers from one general problem: it is possible to detect an application's malfunction in the moment it happens but it still does not help in finding the root cause of the problem. Therefore in general this class of techniques is only useful to verify fulfillment of SLAs from a customer's point of view, but additional techniques have to be used for further analysis in order to detect the reason of a QoS problem. Our studies revealed two basic methods that are applied to monitor from a client's perspective: *synthetic transactions* and *GUI based solutions*. The following paragraphs describe these two approaches in more detail.

Synthetic Transactions: This method uses simulated transactions in order to measure the response time of an application server and to verify the received responses by comparing them to previously recorded reference transactions. Several simulator agents, acting as clients in a network, send requests to the application server of interest and measure the time needed to complete a transaction. In case response time exceeds a configurable threshold or the received server response is incorrect in some way, the agents usually inform the manager by generating events.

As solely synthetic transactions are monitored and not real transactions initiated by actual users, this technique is only useful to take a snapshot of a server's availability, but not to verify the fulfillment of service level agreements. To get measurement data close to actual user experience, the interval between simulated transactions has to be reduced to a minimum. As a consequence the application service could experience serious performance degradation. Further problems arise from agent deployment in large networks.

Currently there are several companies providing either product solutions for performance monitoring using synthetic transactions, like *Geyer & Weinig's GW-TEL INFRA-XS* [15], or offering a service for testing the availability of application services, like *Jyra In-Site* for Web–/E–Commerce Server [11], using this technique.

GUI based solutions: To be able to meter the actual user transactions but to avoid the need for accessing the client application's source code, a new approach was recently developed: As every user request both starts and ends with using/changing a GUI element at the client side (e.g. clicking a web link and displaying the appropriate web page afterwards), simply observing GUI events delivers the needed information about start and end points of user transactions. A software agent installed at client site devices gathers the transaction data of interest from a user's point of view.

The advantages of this technique are that the actually occurring transaction duration is measured and that it can be applied to every application service client. Furthermore, only very little performance impact is caused on the monitored application.

However, we discovered two major problems. First of all, mapping GUI events to user transactions is a difficult task regarding non–standard applications and therefore requires additional effort by the administrator. Secondly, the only agents using these technique known by the authors, are agents by *Candle ETE-Watch* [9] that are currently available only for MS Windows platforms.

2.4. APPLICATION–WIDE MONITORING

As mentioned before, client–based monitoring cannot identify the reason for performance degradation or malfunction of an application. Therefore solutions that monitor both from the client– and from the server–side are necessary. As details about the application and problems within the application cannot be gathered externally, these approaches rely on information supplied by the application itself. Our studies have shown two basic classes that allow application–wide monitoring. These are *application instrumentation* and *application description*. These two classes are described in the following paragraphs in greater detail.

Application instrumentation: Application instrumentation means insertion of specialized management code directly into the application's code. The required information is sent to management systems by using some kind of well–defined interface. This approach can deliver all the service–oriented information needed by an administrator. The actual status of the application and the actual duration of transactions is measured and any level of detail can be achieved. Subtransactions within the user transactions can be identified and measured.

However, application instrumentation is not very commonly used today. This is mainly due to the complexity and thus the additional effort posed on the application developer. The developer has to insert management code manually when building the application. Subtransactions have to be correlated manually to higher–level transactions. As the source code is needed for performing instrumentation, it definitely has to take place during development.

Examples for approaches using application instrumentation are the *Application Response Measurement API (ARM)* [4] jointly developed by HP and Tivoli and the *Application Instrumentation and Control API (AIC)* [5] developed by *Computer Associates*. Both approaches have recently been standardized by the Open Group. ARM defines a library that is to be called whenever a transaction starts or stops. Subtransactions can be correlated using so called correlators. Thus the duration of the transaction and all subordinate transactions can be measured. AIC in contrast was not explicitely developed for performance measurement but might be used in this area as well. It defines an application library to provide management objects that can transparently be queried using a client library. Additionally, a generic management function can be called through the library and thresholds of certain managed objects can be monitored regularly. Both ARM and AIC suffer from all the problems mentioned above and thus are not in wide–spread use today.

Application description: As most of the applications in use today somehow deliver status information but are not explicitely instrumented for management, application description techniques can be used. As opposed to the instrumentation approach, no well–defined interface for the provisioning of management information exists. The description therefore comprises where to find the relevant information and how to interpret it. Examples might be scanning of log files or capturing status events generated by the application.

The major advantage of application description techniques is that it can be applied to legacy applications without requiring access to the source code. It can be done by a third party after application development, while the reasonable approach again is to provide the description by the developer.

Application description faces two major problems: The information available typically is not easy to map to the information needed by the administrator. Especially in the area of performance management typically only little information is available. Moreover monitors are needed to extract the information from the application. Only very little information can be gathered by standard monitors and thus specialized monitors must be developed for every application.

The most prominent representative of application description suited for performance monitoring is the *Application Management Specification (AMS)* [1]. Most other approaches, like the *CIM Application Schema* [7] mainly focus on configuration management. An example for a tool making use of application description is *Tivoli Business System Manager* [14], which reads in AMS based *Application Description Files (ADF)* to learn about the application or business system to be managed.

3. OPEN RESEARCH QUESTIONS

Table 1 summarizes the results of our analysis. As can easily be seen, none of the currently existing techniques completely meets the requirements of service oriented application management. Either they simply cannot deliver the data needed or they suffer from high complexity.

Techniques	Requirements	Service–oriented parameters	Actual user experience	In–depth information	Minimal effort	Minimal impact	Real–time measurement	General applicability
Monitoring of network traffic		o/–	o	–	–	o/+	–	++
Monitoring of system parameters		––	o	o/+	o	++	++	++
Client–based monitoring	Synthetic Transactions	+	––	––	++/+	––	++	++
	GUI–based Monitoring	++	++	––	–	++	++	–
Application–wide monitoring	Application instrumentation	++	++	++	––	+	++	o
	Application description	o	o	o	–	++	+	++

Table 1: Summary

In our opinion application instrumentation is the only reasonable way to overcome the problems of today's application performance management. However, due to the enormous efforts posed on the developer nowadays it is hardly ever used. This immediately leads to a number of research questions that have to be tackled in the future. The following enumeration provides an overview of the most important open research questions concerning application instrumentation:

Instrumentation methodology: A methodology for the developer must be provided in order to alleviate the task of identifying relevant measurement points. Current application techniques simply offer APIs to be called from an application but give no hint about where to actually place the calls in the code. The instrumentation definitely has to take place during application development and thus the instrumentation methodology must be integrated into the software development process.

Automation and tool support: To further alleviate a developer's task a great amount of automation and tool support must be achieved. Therefore, means to facilitate or even automate the instrumentation process must be developed. E.g., in the area of component based application development the code required to measure response time of an individual component might entirely be generated by a development tool.

Correlation of subtransactions: Means to avoid the cumbersome correlation of subtransactions to their parent transactions are desperately needed. The idea of transporting unique transaction identifiers through the application as parameters is awkward to say the least. In some cases (e.g., component bases development, third party instrumentation) this approach even makes an instrumentation nearly impossible. By correlating transactions, e.g., using the identifiers of the underlying control flows, this could be simplified by far.

Integration with remaining existing techniques: Finally, a tighter integration of application instrumentation with the existing techniques is required in order to deliver an exhaustive overview of the status of the application to be monitored.

4. CONCLUSION

The paper focused on evaluating the different approaches currently used for performance monitoring of distributed applications. Therefore a classification and analysis of available approaches has been done. Overall result was, that current techniques are not sufficient to solve today's service management problems. The paper concludes with open research questions in the area of application performance management that – in our opinion – are most important to be tackled in the forthcoming years.

Our current work focuses on the research questions raised above. Especially in the area of component based application development some promising prototypes are under development that might lead to a high degree of instrumentation automation. In the area of automated transaction correlation we already have achieved some first substantial progress which is about to be published soon.

Acknowledgment

The authors wish to thank the members of the Munich Network Management (MNM) Team for helpful discussions. The webserver of the MNM Team is located at `http://wwwmnmteam.informatik.uni-muenchen.de`.

References

[1] Application Management Specification. Version 2.0, Tivoli Systems, 1997.

[2] Apptitude. MeterFlow Network Decision Data Engine. Technical White Paper, January 2000.

[3] N. Brownlee, C. Mills, and G. Ruth. RFC 2063: Traffic flow measurement: Architecture. RFC, IETF, January 1997.

[4] Application Response Measurement (ARM) API . Technical Standard C807, The Open Group, July 1998.

[5] Application Instrumentation and Control (AIC) API, Version 1.0. Technical Standard C910, The Open Group, November 1999.

[6] Compuware. Ecoscope — analyzing networked application performance, 2001.

[7] DMTF Application Working Group. Application MOF Specification 2.5. CIM Schema, Distributed Management Task Force, December 2000.

[8] M. Garschhammer, R. Hauck, H.-G. Hegering, B. Kempter, M. Langer, M. Nerb, I. Radisic, H. Roelle, and H. Schmidt. Towards generic Service Management Concepts – A Service Model Based Approach. In *Proc. of the 7th Int. IFIP/IEEE Symposium on Integrated Management*, May 2001.

[9] Hurwitz Group. Candle captures the "user experience". Technical White Paper, September 1998.

[10] Hewlett–Packard Company. HP Management Software Products, 2001.

[11] Jyra. Jyra In–Site@PSINet, 2000.

[12] C. Krupczak and J. Saperia. RFC 2287: Definitions of system-level managed objects for applications. RFC, IETF, February 1998.

[13] V. Paxson, G. Almes, J. Mahdavi, and M. Mathis. RFC 2330: Framework for ip performance metrics. RFC, IETF, May 1998.

[14] Tivoli Systems, Inc. Tivoli Business System Manager, 2001.

[15] Geyer & Weinig. Portofolio – INFRA-XS, 2000.