INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Systementwicklungsprojekt

Evaluierung von Werkzeugen zur Antwortzeitüberwachung bei der DeTeSystem

Bearbeiter:	Michael Fischer
Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Dipl. Inf. Rainer Hauck Dipl. Inf. Igor Radisic Helmut Dürr (DeTeSystem)

INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Systementwicklungsprojekt

Evaluierung von Werkzeugen zur Antwortzeitüberwachung bei der DeTeSystem

Bearbeiter:	Michael Fischer
Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Dipl. Inf. Rainer Hauck Dipl. Inf. Igor Radisic Helmut Dürr (DeTeSystem)

Abgabetermin: 7. März 2001

Zusammenfassung

Der nachfolgende Projektbericht behandelt das Thema Antwortzeitüberwachung auf Applikationsebene, wobei sowohl auf die konzeptionellen wie auch auf praktische Aspekte eingegangen wird. Neben einer Darstellung der theoretischen Konzepte zur Antwortzeitüberwachung, enthält der Bericht einen Überblick über derzeit erhältliche Produkte sowie deren praktische Evaluierung. Als Produkte sind hierbei *Infra-XS* von der Firma *Geyer und Weinig*, *Etewatch von Candle*, sowie *Firehunter von Agilent/HP* und *ARM 3.0 der CMG Working Group* zu nennen. *Etewatch* stellte sich als Tool mit dem innovativsten Ansatz vor, vermittelte jedoch durch inkorrekte Messergebnisse ein gewisses Maß an Unzuverlässigkeit. Das Performance Tool *Firehunter*, welches Antwortzeiten mittels simulierter Anfragen misst, machte einen guten Eindruck, fiel aber dennoch durch die sehr hohen Systemanforderungen auf. *Infra-XS* von Geyer & Weinig, ein äußerst komplexes Tool, benutzt eine Kombination aus simulierten Anfragen und Netzüberwachung, um Rückschlüsse auf die Kommunikationszeiten von End-zu-End Beziehungen zu erhalten. Dabei fiel besonders die hohe Komplexität der Software auf, die zwar den professionellen Charakter des Produkts unterstützt, die Bedienung und Installation jedoch nur durch geschultes Fachpersonal der Herstellerfirma selbst ermöglicht. Das letzte getestetes Produkt, ARM 3.0, wartet in seiner neusten Version mit Java Funktionalität und einem erweiterten Befehlsrepertoire auf. Trotz herausragender Eigenschaften spricht extrem hoher Implementierungsaufwand gegen einen Einsatz.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Heutige IT-Landschaften bestehen zunehmend aus großen und komplexen Netz- und Systemarchitekturen, deren reibungsloser Betrieb die IT-Dienstleister vor große Herausforderungen stellt. Gerade im Bereich E-Commerce werden zudem noch hohe Anforderungen an kalkulierbare und kontrollierbare Leistungsfähigkeit und Verfügbarkeit gestellt. Schnelle Reaktionszeiten sind deshalb unabdingbar in geschäftskritischen Anwendungen und sind ausschlaggebend für die Anwenderzufriedenheit und dem daraus resultierenden wirtschaftlichen Erfolg eines Unternehmens. Für die Betreiber von E-Commerce Seiten ist es aufgrund der Komplexität der Infrastruktur zwischen Web-Server und Benutzer PC oftmals schwer nachvollziehbar, ob sie die geforderten Antwortzeiten einhalten können. Um diesem Problem Rechnung zu tragen, müssen geeignete Messverfahren gefunden werden, welche eine Überwachung von End-zu-End-Verbindungen aus der Kundensicht möglich machen, damit Servicelevel-Nachweise durchgeführt werden können. Momentan führen Unternehmen lediglich Ressourcenüberwachung ihrer IT-Systeme durch. Z.B. wird durch ICMP Anfragen (Ping) die Verfügbarkeit von Ressourcen festgestellt oder durch einfache Performance Tools die Auslastung des Netzes betrachtet. Diese Messverfahren alleine sind für die Einhaltung von Service Level Agreements als ungeeignet einzustufen, da im Zentrum der Betrachtung die zu erbringenden IT-Serviceleistungen, wie Transaktions-, Verarbeitungs-, Netz-, und Host-Antwortzeiten stehen müssen. Diese Problematik hat zu unterschiedlichsten Ansätzen in der Messung von IT-Serviceleistungen geführt und verschiedenste Produkte hervorgebracht. Diese Arbeit soll einen Überblick über die momentan erhältlichen Produkte geben und deren Stärken und Schwächen in der Praxis aufzeigen. Dazu werden neben den Produkten selbst, die verschiedenen Konzepte einer Antwortzeitüberwachung erläutert und nach Kriterien wie Zuverlässigkeit, Aufwand und Qualität beurteilt. Anschließend wird zu jedem konzeptionellen Ansatz ein Produkt vorgestellt und einem praktischen Test unterzogen.

1.2 Bewertungskriterien

Der folgende Krtierienkatalog soll helfen die Konzepte und Produkte in einem vernünftigen Rahmen zu bewerten und einzuordnen. Im Bereich der Antwortzeitüberwachung gibt es eine Reihe von Kriterien deren Erfüllung wünschenswert oder sogar notwendig sind.

- **Aussagekraft der Messung**
  Eine wichtige Eigenschaft der Antwortzeitüberwachung ist die Aussagekraft der einzelnen Messungen. Um genaue und zugleich relevante Ergebnisse zu erzielen, sollten die Messungen so nahe wie möglich an der zu überwachenden Applikation ansetzen. Es ist nur zum Teil sinnvoll auf repräsentative Messungen eines anderen Standorts zurückzugreifen und sie dann auf die eigene Location zu übertragen. Je näher man deshalb an der eigentlichen Anwendung misst, desto aussagekräftiger sind die Resultate.
- **Installationsaufwand und Systemanforderungen**
  Weitere Bewertungskriterien sind Installationsaufwand und Voraussetzungen an Hard- und Software. Leichte und einfache Installation und Wartung sowie genügsame Systemanforderungen sollten gewährleistet werden. Komplizierte Installationsroutinen oder notwendige Änderungen an bestehender Software erfordern hohen Arbeits- und Kostenaufwand und sind deshalb zu vermeiden.
- **Einsatzkosten**
  Unter Einsatzkosten sind Aufwendungen für den Erwerb der eigentliche Software und notwendiger Hardware zu verstehen. Besonders für große Unternehmen mit sehr vielen Computern entscheidet häufig der Kostenfaktor über Anschaffung und Einsatz.
- **Detaillierungsgrad der Messdaten**
  Eine Transaktion besteht normalerweise aus verschiedenen zusammengesetzten Subtransaktionen, wobei das Messen der Gesamttransaktion der Grundgedanke der Antwortzeitmessung darstellt. Sind Überwachungssysteme jedoch zusätzlich in der Lage, neben Gesamttransaktionszeiten auch einzelne Subtransaktionszeiten zu erfassen, hätte dies eine Erhöhung der Diagnosefähigkeit zur Folge. Damit sind wesentlich schnellere Störungsanalysen möglich, was in einem kleinen Beispiel verdeutlicht werden soll. Bestandteile eines Webseitenaufrufes bestehen z.B. vereinfacht gesehen aus DNS Auflösung und Seitendownload. Kann eine Transaktionsmessung nun dahingehend aufgelöst werden, neben der Gesamtzeit der Transaktion (komplettes erscheinen der Seite) auch die Einzelzeiten der Subtransaktionen (Namensauflösung und Download) zu bestimmen, würde dies eine Lokalisierung von Fehlern erleichtern.

Diese Kriterien sollen den groben Rahmen der Evaluierung bilden und werden am Schluss anhand der einzelnen Produkte nochmals zusammengefasst.

Kapitel 2

Konzepte zur Antwortzeitüberwachung

Das Ziel des Applikation Response Measurement ist es, Antwortzeiten die der Endbenutzers bei der Arbeit mit vernetzten Anwendungen erfährt, zu messen und auszuwerten. Wichtiger Verwendungszweck sind z.B. Erhalt von Nachweisen für die Einhaltung von Service-Level-Agreements. Dies erfordert Zeitmessungen, die die Dauer zwischen einem Transaktionswunsch des Anwenders und dem dazugehörigen sichtbaren Resultat erfassen. D.h. es ist nicht sinnvoll mit Methoden wie z.B. *Ping* diese Zeiten zu messen, da das Ergebnis nur Aufschluss über die Verfügbarkeit von Resourcen gibt aber nichts über die Qualität eines gewünschten Dienstes aussagt. So wird die Notwendigkeit deutlich, entsprechende Meßmethoden zu finden, die sich an diesen Vorgaben orientieren.

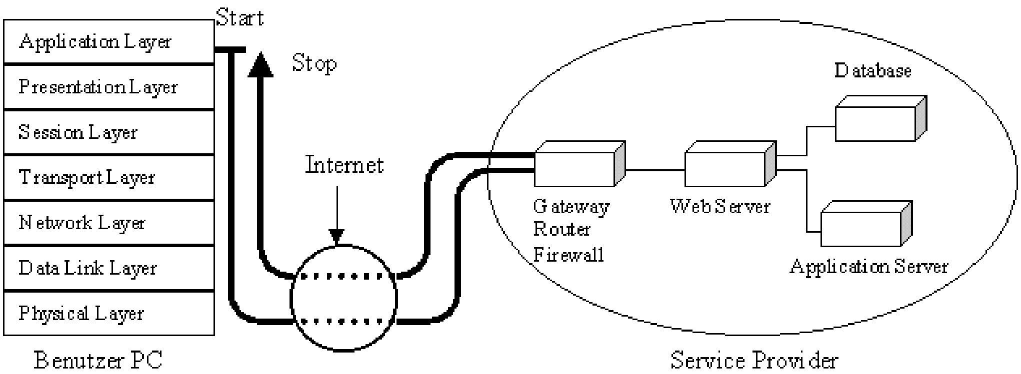


Abbildung 2.1: Application Response Measurement Schema

Abb. 2.1 macht die gewünschten Messpunkte nochmals deutlich. Die hier gezeigte Grafik erläutert einen möglichen Transaktionsweg vom Benutzer zum Service Provider. Die Start und Stop Symbole markieren eine Zeitmessung die eine Transaktion braucht, um vom PC des Anwenders zum Service Providers zu gelangen, dort ausgewertet und bearbeitet zu werden und zurück zu kommen. Diese Zeit beinhaltet neben der Transportzeit auch die Bearbeitungszeit mittels Web Server, Database Server oder Application Server. Das bedeutet, es soll die Dauer zwischen Aufgabe der Benutzeranfrage und Erhalt der Antwort (meist sichtbar) gemessen werden. Zeitmessung dieser Art spiegeln die *Wartezeiterfahrung* des Anwenders wieder und erlaubt so Aussagen über

Service-Level-Agreements zu machen. Nicht zuletzt bedeutet dies auch ein gewisses Mass an Managementfunktionalität ([HAN99], da mit derartigen Überwachungssysteme, neben Antwortzeiten auch Zustand und Funktionsfähigkeit überwacht werden können.

Im folgenden werden nun verschiedene Konzepte erläutert, wie eine solche Messung praktisch durchgeführt werden kann.

2.1 Eingabeoberflächenbasierte Auswertung

Eine der intuitivsten Methoden eine Transaktionsmessung durchzuführen ist eine Auswertungsmethode basierend auf der Eingabeoberfläche. Eingabeoberfläche bedeutet hierbei die Kommunikationsschnittstelle zwischen Benutzer und Anwendung. Möchte der User eine Transaktion in Gang setzen, so benutzt er eine Eingabeoberfläche, um seine Transaktion zu spezifizieren und zu starten. Dieser Start kann z.b. ein *Mausklick* sein oder eine Eingabe, die mit *Return* abgeschlossen wird. Das Ende einer Transaktion wird meistens durch eine Veränderung des Bildschirmaufbaus z.B. durch eine Status Rückmeldung beispielsweise in Form einer Zeichenfolge wie *Complete* angezeigt. Die Eingabeoberflächenbasierte Auswertung benutzt diese Merkmale, um Start- und Endpunkt einer Transaktion festzustellen und berechnet die Zeit dazwischen. Generell können Funktionsaufrufe wie *START*, *STOP* oder *ABBRUCH* aufgefangen und schließlich in Antwortzeiten reflektiert werden.

Ein Beispiel hierzu: Ein Benutzer gibt eine URL in seinen Browser ein und drückt *Return*, um die Seite anzeigen zu lassen. Das Ereignis *Return drücken* kann als Transaktionsstart verstanden werden und leitet damit den Beginn der Zeitmessung ein. Ist die Seite vollständig geladen, erkennbar z.B. an einem verschwundenen Fortschrittsbalken, ist dies als Ende der Transaktion zu bewerten und die Zeitmessung stoppt. Damit lassen sich End-to-End Antwortzeiten erzeugen, die die reine "Wartezeit-"Erfahrung des Benutzers widerspiegelt.

Vorteile:

Ausschlaggebender Vorteil dieser Methode ist der geringe Aufwand im Einsatz. Da sich diese Methode ausschließlich als Beobachter der Eingabeoberfläche der Anwendung versteht, sind keine Eingriffe in den Programmkern oder der Treiber der eigentlichen Anwendung nötig, um eine Überwachung durchführen zu können. Transparenz ist ein weiterer Grund, der für den Einsatz dieser Methode spricht. Die überwachte Anwendung benötigt keinerlei Informationen über die Messoftware, noch müssen Veränderungen in vorhandenen Einstellungen durchgeführt werden. Auch die Überwachung direkt an der eigentlichen Applikation, bedeuten einen großen Vorteil dieses Konzeptes. Durch die direkte Messung beim Endbenutzer selbst, spiegeln die so gewonnene Daten ein exaktes Bild der Wartezeit-Erfahrung des Anwenders wieder. Dadurch ergeben sich aussagekräftige und genaue Ergebnisse des Antwortverhaltens aus der Benutzersicht.

Nachteile:

Ein Nachteil dieser Messmethode ist die Notwendigkeit von Kollektoren. Kollektoren führen die Überwachung der Anwendung durch und erfassen Transaktionszeiten. Das bedeutet, jedes Rechensystem welches mit in die Antwortzeitüberwachung mit eingeschlossen werden soll, muss mit einem Kollektor ausgestattet werden. Ein weiterer nachteiliger Punkt, ist der fehlende Detaillierungsgrad von Messungen. Vielfach ist durch dieses Verfahren nur Start und Ende einer Transaktion ermittelbar. Zeiten für Zwischenschritte einer komplexen Transaktion sind dabei nicht aufzulösen und machen gezielte Problemlösungen bei zu hohen Antwortzeiten unmöglich. Problematisch verhält es sich auch bei Transaktionen die keine augenscheinliche Veränderung an der Eingabeoberfläche verursachen. Denkbar wäre z.B. das Abschicken von E-Mails. Das erfolgreiche Verschicken von

E-Mails erzeugt meistens keine sichtbaren Änderungen auf der Eingabeoberfläche und erschwert damit die Erkennung des Endes solcher Transaktionen.

2.2 Simulation von Anfragen

Eine weitere Methode, Antwortzeitüberwachung auf Applikationsebene durchzuführen, ist die Simulierung von Anfragen. Ein dediziertes System führt vordefinierte Anfragen auf Services durch und misst ihre Performance. Spezielle, mit Messroutinen ausgestattete Programme übernehmen dabei die Rolle des Benutzers, indem sie automatisiert und selbständig Transaktionen durchführen. Diese Transaktionen oder auch simulierte Anfragen können unterschiedlichster Natur sein, wie z.B. Webseitenaufruf, Filetransfer, Namensauflösung per DNS, Verschicken von E-Mails usw. Die Ergebnisse werden gesammelt und an eine zentrale Verwaltungsstelle geschickt. Zusätzlich zu diesen Automatisierten Abläufen kann auch eine Benutzereingabe simuliert werden. Das bedeutet, ein gewünschtes Benutzerverhalten (z.b. Aufruf verschiedener Webseiten mit anschließendem Filedownload) wird aufgenommen und anschließend vom System in gleicher Weise wiedergegeben und gemessen.

Vorteile:

Als Vorteile bei dieser Messmethode, sind geringe Installationszeiten sowie Wartungskosten zu nennen. Bei diesem Verfahren werden nur einige wenige dedizierte Systeme mit Messoftware ausgestattet. Damit beschränkt sich eine Installation und Wartung auf diese wenigen Systeme, was eine Kosten- und Zeitersparnis mit sich bringt. Ein weitere Punkt der hier zu nennen wäre ist die Erhöhung des Detaillierungsgrades der Transaktionsmessungen. Während bei Eingabeoberflächenbasierter Messung lediglich Start und Endpunkt einer Transaktion festgestellt werden können, ist es hier möglich durch proprietäre Software auch Teiltransaktionszeiten einer größeren komplexen Transaktion zu messen. Um ein Beispiel anzuführen, ein FTP Download könnte z.b. in Connect Time und eigentlichen Download aufgesplittet werden. Damit ist neben der Überwachung von Qualitätsvereinbarungen auch die Möglichkeit gegeben auftretende Störungen schnell und gezielt zu beheben.

Nachteile:

Wie oben erwähnt, werden nur einige ausgewählte Rechensysteme als Messysteme eingesetzt, die anschließend automatisch nach einem vordefinierten Schema Messungen durchführen. Damit ergeben sich folgende Probleme. Zum einen werden Transaktionsmessungen nur stichprobenartig durchgeführt, was das Erkennen von Störungen verzögert. Ein gutes Beispiel hierfür ist die Überwachung eines Webservers. Theoretisch ist eine Überwachung aller verfügbaren Webseiten zwar möglich, aus Rentabilitätsgründen beschränkt man sich jedoch auf einen kleinen repräsentativen Teil aller zur Verfügung stehenden Seiten und Dateien. Kommt es zu Problemen, z.b. Seiten stehen nicht mehr zur Verfügung oder sind nur sehr schwer zu erreichen, kann dies, aufgrund der Stichprobentests, nur mit erheblichen Zeitverzögerungen festgestellt werden. Ein weiterer nachteiliger Punkt ist die Aussagequalität der erzielten Ergebnisse. Aussagen aus den gewonnenen Messdaten können nur als grober Schätzwert für umliegende Endbenutzer Systeme gelten. Grund dafür ist zum einen die unterschiedliche Netzanbindung von Messystem und Endbenutzersystem. Die Messtation ist z.B. alleine an einem 100Mbit Knoten mit dem Netz verbunden, während die Endbenutzersysteme sich zu mehreren einen 10Mbit Knoten teilen müssen. Desweiteren verhindert die unterschiedliche Ausstattung mit Hard- und Software der Rechner die exakte Übertragung der Messergebnisse auf andere Rechner.

2.3 Überwachen des Netzverkehrs

Eine weiteres Konzept zur Ermittlung von Transaktionszeiten, ist die Überwachung des Netzverkehrs. Messtationen werden in den Übertragungsweg zweier Netzknotenpunkten gehängt, wie z.b. eine Backboneverbindung von Kundennetz zum ISP. Diese Messtationen, auch Probes genannt, verfolgen den Netzverkehr und analysieren dabei Pakete und verwendete Ports. Anhand dieser Informationen werden Rückschlüsse auf das Antwortzeitverhalten der jeweiligen Anwendungen gemacht. Frame-Header Informationen wie IP Adresse, Port Nummer und Anwendungsspezifische Daten (http Requests, Format spezifisch), werden überwacht und in Relation zu Anwendungen gebracht.

Vorteile:

Dieses Konzept hat Vorteile ähnlich der simulierten Anfragen. Zeit und Kosten in der Einrichtungsphase erspart man sich auch hier durch den Wegfall von Instrumentierung von Anwendungen und Installation von Kollektoren. Da die Überwachung ausschließlich von einigen wenigen Geräten durchgeführt wird, ist zudem schnelle und leichte Wartung sowie Einrichtung gewährleistet. Werden Probes sowohl auf Benutzerseite wie auch auf Provider Seite aufgestellt, ist eine Korrelierung der Daten möglich und auch sinnvoll. Dies erlaubt das Laufzeitverhalten der Paketströme in den einzelnen Teilstrecken, wie z.B. Benutzerbereich, ISP und Serverumgebung zu messen. Als Resultat erhält man eine wesentlich höhere Auflösung für Problemanalysen, die zu schnelleren und effektiveren Lösungsansätzen führt.

Nachteile:

Nachteile dieses Konzepts sind zum einen die Analysekomponenten (Probes) selbst. Dadurch dass eine Applikationsmessung auf Basis ihres Netzwerkstromes stattfindet, müssen einige grundlegende Eigenschaften beachtet werden. Eine Messung findet anhand der Analyse des Datenstroms auf der Netzwerkebene statt. Dahingehend muss auch eine geeignete Positionierung der Probe erfolgen. D.h., um eingehenden und ausgehenden Netzverkehr mehrerer Systeme zu analysieren, ist die Probe in innerhalb einer relevanten Backboneverbindung aufzustellen. Diese Verbindungspunkte sind meistens Netzübergangspunkte mit Hochgeschwindigkeitsverbindungen. Um nun einerseits Pakete aus diesem Strom herauszulesen und auszuwerten und andererseits den Netzstrom so wenig wie möglich zu beeinflussen, sind hohe Anforderungen an die Probes zu stellen. Probes die diesen Anforderungen gerecht werden wollen, müssen eine leistungsstarke Kombination aus Hard- und Software aufweisen, die teilweise extrem teuer sein kann. Zum anderen gestaltet sich die Beobachtung von Paketen und ihre Zuordnung zu Applikationen sehr schwierig. Alleine aufgrund der großen Anzahl von Anwendungen, die sich auf der Netzebene jeweils mit den unterschiedlichsten Protokollen unterhalten, macht eine Verfolgung und Zuordnung jedes einzelnen Paketes äußerst kompliziert. Eine exakte Überwachung aller möglichen Protokolltypen ist deshalb nur sehr schwer zu realisieren und stellt damit einen weiteren Minuspunkt dieses Konzeptes dar.

2.4 Instrumentierung

Als letzte Methode sei die eigentliche Instrumentierung der Software genannt. Dieses Verfahren, auch unter dem Namen ARM (Application Response Measurement) bekannt, ermöglicht Antwortzeitmessungen mittels expliziter Markierung von Transaktionsfunktionen innerhalb des Sourcecodes der eigentlichen Anwendung. Diese Markierungen legen Start- und Endpunkte einer Transaktion fest und machen so eine Zeitmessung erst möglich. API Aufrufe dieser Messpunkte werden anschließend von geeigneten Agenten aufgefangen und in verständliche Daten, sprich

Transaktionszeiten, umgewandelt.

Vorteile:

Hauptvorteil dieser Methode ist, dass die Messung direkt an der gewünschten Anwendung durchgeführt wird. Die Überwachung geschieht direkt aus der Sicht des Benutzers und reflektiert so seine persönliche Wartezeiterfahrung. Präzise und repräsentative Aussagen über Antwortzeitverhalten sind das Ergebnis einer derartig *armierten* Anwendung. Gemessen werden kann dabei nicht nur die Gesamtantwortzeit einer Transaktion, sondern auch einzelne Subtransaktionen. Z.B. kann eine Webseitenanfrage in zwei Komponenten aufgeteilt werden. Einmal in der DNS Auflösung der eingegebenen URL, zum zweiten in die eigentliche Übertragung der Seite. Diese Detailinformationen machen sich folglich auch in der Reaktionszeit bei Problemfällen bemerkbar, auf die dadurch wesentlich schneller und zielstrebiger eingegangen werden kann. Durch Einsatz von ARM Komponenten sowohl in Client und Server Bereich ergibt sich eine weitere günstige Eigenschaft. Transaktionen können korreliert werden. D.h. Transaktionen die eine weitere Transaktion auslösen, können als zusammengehörig identifiziert werden. Ermöglicht wird dies, indem sog. Identifikatoren und Korrelatoren mitgeschickt werden, welche die Aufgabe haben Transaktion ausführlicher zu beschreiben. Dies erlaubt ein Gesamtbild einer komplexen Transaktion, mit den entsprechenden Teilzeiten, zu kreieren.

Nachteile:

Obwohl Instrumentierung sehr viele positiven Eigenschaften besitzt, ist der Einsatz mit einigen Problemen verbunden. Hauptgrund hierfür ist der extrem hohe Implementierungsaufwand. Kaum eine derzeit erhältliche Software ist mit ARM API-Aufrufen ausgestattet oder verfügbar. Darüber hinaus benutzen Firmen meist eigene, individuell an ihre Bedürfnisse angepasste Software. Eine nachträgliche Aufrüstung dieser Software müsste auf Code Ebene stattfinden und würde zu einem sehr großen Zeit- und Kostenaufwand führen. Oft ist auch eine Umrüstung großer verteilter Anwendungen, aufgrund ihrer Komplexität und der damit verbundenen nicht absehbaren Konsequenzen, einfach nicht möglich. Auch der Einsatz von Korrelatoren zur Identifikation von zusammengehörigen Transaktionen, ist mit hohem Aufwand verbunden. Eine Korrelierung von Transaktionen benötigt neben der clientseitigen ARM API Erweiterung zusätzlich die Ausstattung der Server Systeme mit ARM API-Aufrufen. Dieser immense Implementierungsaufwand spricht daher stark gegen den Einsatz von ARM.

Es sei noch gesagt, dass lediglich eine oberflächliche Betrachtung der Konzepte zur Antwortzeitmessung hier durchgeführt wurde und eine weit ausführlichere Behandlung dieser Thematik, in der Arbeit von Hauck und Radisic ([HR00]) zu finden ist.

Kapitel 3

Produkte zur Antwortzeitüberwachung

3.1 Etewatch V1.2 von Candle

Ein aktuelles Produkt, dass sich unter das Konzept *Eingabeoberflächenbasierte Überwachung* einordnen lässt, ist *Etewatch* von Candle. *Etewatch* fängt Oberflächenspezifische Eingaben ab, um sie in Antwortzeiten zu reflektieren. Abb. 3.1 erläutert kurz den generellen Aufbau einer ETEWATCH Umgebung. Kollektoren (hier links) auf der Clientseite messen die Antwortzeiten und geben sie an einen Manager Client weiter, der sich auf der selben Maschine befindet. Diese Daten werden kollektiv an einem Punkt, dem Manager Server, gesammelt und können mit Hilfe eines Reporting Tools individuell ausgewertet werden.

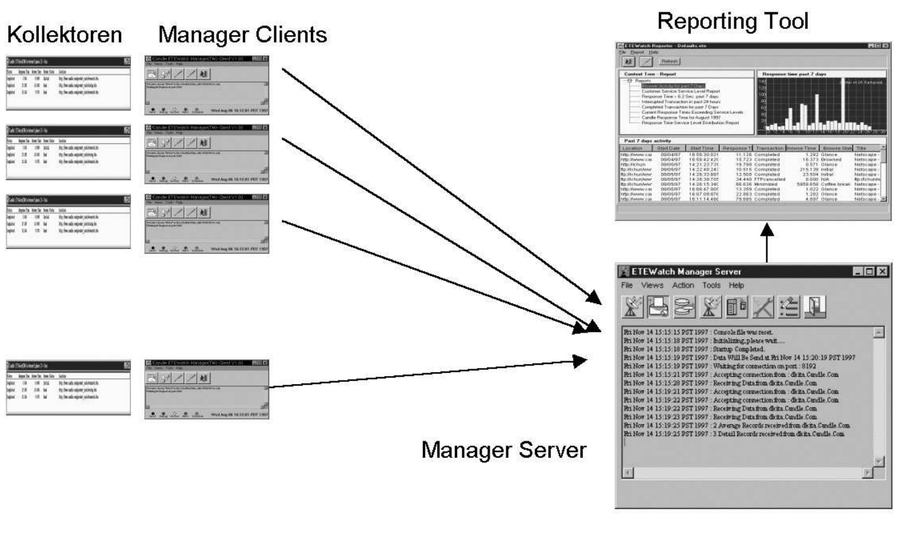


Abbildung 3.1: Beispielaufbau einer Etewatch Überwachung (nach [Canb])

3.1.1 Systemvoraussetzungen

Nachfolgende Tabellen zeigt die herstellerspezifischen Systemanforderungen für ein Etewatch Überwachungssystem.

Client Workstation with Manager Client and Collector	
OS	Windows NT Service Pack 4, Win95, Win98 mit Winsock2
CPU	Minimum 486/66 MHz
Disk	Ca. 10 MB
Memory	Minimum 32 MB
Software	JRE oder JDK version 1.1.6

Manager Server and Reporting Tool	
OS	Windows NT Service Pack 4, Win95, Win98 withWinsock2
	AIX 4.2 or 4.3
	Solarix 2.5, 2.6, 2.7
	HP-UX 10.20, 11.0
CPU	Minimum P90 MHz od. ähnliches bei Unix
Disk	Ca. 10 MB und zusätzlicher Speicher für Datenbank Dateien
Memory	Minimum 32 MB
Software	ODBC od. JDBC Treiber für Datenaustausch mit anderen Datenbanken
	JRE oder JDK Version 1.1.6
	Verbindung mit Server LOG Files für Reports (Drive Mapping)

3.1.2 Komponenten von Etewatch

3.1.2.1 Kollektor

Der Kollektor ist das Herzstück des Produkts. Mit ihm können Oberflächenevents von verschiedensten Anwendungen beobachtet und in verständliche Daten verwandelt werden. Der Kollektor ermöglicht eine Echtzeitanzeige der gesammelten Daten, wobei er sie in Form einer Tabelle präsentiert:

- Status: Momentaner Zustand der überwachten Transaktion
- Time: (Web Browser only) Uhrzeit des Transaktionsaufrufes
- Download Time (Web Browser only): Zeit zwischen Verbindungsaufbau und download der Webseite.
- Total Time: Gesamte Zeit zwischen Benutzeranfrage und Erledigung der Transaktion
- Browse Time: Verbrachte Zeit auf Seite
- Interest (nicht mit IE 4,5) : Angabe wie oft auf der Seite gescrollt wurde
- Transaction: für Webbrowser URL, Seite und File eines Webbrowser Downloads, Name oder IP für nicht Browser abhängige Applikationen
- Title: Fenstertitel der Applikation
- Executable Name: Filename der überwachten Applikation

Eine Besonderheit des Kollektors ist seine Fähigkeit gesammelte Daten in ein ARM 1.0 oder ARM 2.0 kompatibles Format zu bringen. Diese Daten können dann anschließend von ARM fähigen Management Stationen ausgewertet werden. Der Kollektor kann auf zwei verschiedene Arten eingesetzt werden. Zum einen als Hintergrund Task ohne manuelle Eingreifmöglichkeit oder als normaler Betriebssystem Task, mit einem sichtbarem Ausgabefenster für gesammelte Daten. Dieses Ausgabefenster wird auch als Personal Viewer bezeichnet, mit dem das Sammeln von Antwortzeiten in Echtzeit verfolgt werden kann. Außerdem ist zu beachten, dass verschiedene Applikationen jeweils einen eigenen Kollektor benötigen, der installiert und gestartet werde muss.

3.1.2.2 Manager Client

Um Messdaten an eine zentrale Stelle zur Verarbeitung und Archivierung weiterzuverschicken, ist zusätzlich zum Kollektor ein Manager Client nötig. Die Installation des Manager Clients muss auf der gleichen Maschine erfolgen auf der auch der Kollektor installiert wurde. Der Client sammelt die ermittelten Daten des Kollektors und schickt sie weiter an einen Manager Server. Der Manager Client kann durch geeignete Einstellung als Filter und intelligente Übermittlungsstation angesehen werden. Bereits im Manager Client lassen sich erfasste Daten durch Filter vorsortieren und entsprechend behandeln. Zusätzlich werden die zur Übertragung bereitstehende Daten komprimiert und in individuell wählbaren Intervallen an den Server geschickt. Damit wird Netzbelastung minimiert um Auswirkungen auf andere Applikationen zu verringern. Der Manager Client ist aber trotzdem in der Lage, bei auftretenden Ausnahmefällen, diese sofort an den Server weiterzuleiten, da er eine Echtzeitüberwachung der gesammelten Daten durchführt. Die Übermittlung kann auf unterschiedliche Weise abgewickelt werden. Informationspakete können sowohl auf TCP Verbindungen, wie auch per E-Mail zum Server gelangen.

3.1.2.3 Manager Server

Der Etewatch Manager Server stellt eine zentrale Datenbank der gesammelten Daten dar. Ihm liegt eine Java Object Database zugrunde die auch von anderen Datenbanken übernommen werden kann, sofern JDBC Treiber dafür zur Verfügung stehen. Der Manager Server bietet darüber hinaus die vollständige Kontrolle über die Manager Clients. Der Austausch von Daten erfolgt nach dem Push-Pull-Modell. D.h. neben dem selbständigen verschicken von Daten vom Client zum Server, kann die Server Station die Client Station aktiv auffordern Daten zu übermitteln. Zusätzlich können Client Einstellungen, wie Definitionen von Ausnahmebehandlungen, Sammelintervalle oder andere Client Kontrollen vom Server aus vorgenommen werden.

3.1.2.4 Reporter

Der Reporter bietet die Möglichkeit, gesammelte Daten aufzubereiten und zu visualisieren. Er besteht aus einer drei geteilten Oberfläche mit Navigation, Graphik und Details (siehe Abb. 3.1). Auf der Navigationsoberfläche wählt man den gewünschten Report aus, dieser wird grafisch im zweiten Teilfenster aufbereitet, während im dritten Fenster Details aufgelistet werden. Wie der Manager Server und Client, liegt auch der Reporter in Java vor und kann von jeder beliebigen Plattform aus eingesetzt werden.

3.1.3 Handhabung

Die Benutzung des Tools ist sehr intuitiv und einfach. Nach der Installation und der Aktivierung des Kollektors sind weder eine gesonderte Behandlung der überwachten Anwendung nötig, noch Einstellungen dergleichen vorzunehmen. Ein weiterer Pluspunkt ist die Tatsache dass Manager

Client/Server und Reporter in Java programmiert wurden und eine Plattform Unabhängigkeit garantieren. Lediglich der Kollektor ist auf reine Windows Systeme beschränkt.

3.1.4 Leistung

Etewatch wird in der Version 1.2 auf einem Windows NT 4.0 (Service Pack4) Rechner einem kurzen praktischen Test unterzogen. Dabei sollen einige Testtransaktionen eines Webbrowsers gemessen und auf Korrektheit überprüft werden. Die Testtransaktionen beinhalten das Messen von Seitenaufbau und Connect Time, Verhalten bei unterschiedlichen Protokollen (HTTP, FTP Anfrage), Erkennen von Abbruchbedingungen bei Connect und Download, sowie korrekte Auswertung bei Fehlerzuständen , z.B. Seite kann nicht gefunden werden.

Zuerst zur Arbeitsweise allgemein. Wird eine URL im Browser eingegeben und RETURN gedrückt oder eine Adresse aus den Favoriten gewählt und mit der Maus angeklickt, übernimmt der Kollektor die Adresse und startet eine Messung. Ist die Seite vollständig im Browser Fenster aufgebaut markiert der *Etewatch* Kollektor diese Messung als beendet und aktualisiert seine bisherigen Daten. Er unterscheidet dabei zwischen HTTP und FTP Anfragen und beschreibt sie auch dementsprechend. Für jede Transaktion wird die Dauer, die Zielseite, die benutzte Anwendung und der Status (Complete, Interupted, oder Canceled) erfasst. Zusätzlich wird bei einem Seitenwechsel die Verweildauer auf der vorherigen Seite festgehalten.

Bei vorsichtiger Bedienung des Explorers, d.h. keine spontanen Abbrüche oder Seitenanfragen während Explorer noch mit der vorherigen beschäftigt ist, protokolliert der Kollektor einwandfrei mit und liefert exakte, repräsentative Messungen des Applikationsverhaltens.

Leider musste festgestellt werden, dass bei intensiver Benutzung des Internet Explorers, Fehlmessungen in den Protokollen des Kollektors auftauchten. Diese werden im folgenden näher ausgeführt.

Die Hauptursache für die Fehlinterpretation von verschiedenen Transaktionszeiten liegt an der fehlerhaften Erkennung des Endes einer Transaktion. Diese Software erkennt ein Ende einer Transaktion am *Refresh* des Ausgabefensters. D.h. Windows hat die Eigenschaft, nach einem vollständigen Transaktionszyklus das Ausgabefenster komplett zu aktualisieren und erledigt das mit einem Fenster '*Refresh*'. Der Kollektor nutzt diese Eigenschaft aus um das Ende einer Transaktion zu markieren.

Wird dieser '*Refresh*' manuell ausgeführt, was bei Windows Systemen möglich ist, nimmt *Etewatch* fälschlicherweise an, dass die Transaktionsanforderung erfolgreich beendet wurde. Hierzu ein Beispiel.

Wie in Abb. 3.2 deutlich zu sehen, versucht der Browser eine Verbindung zu Seite *www.tomorrow.de* aufzubauen. Da sich der Versuchsrechner aber in einer *abgeschlossenen* Umgebung befindet und keinen Zugang ins Internet besitzt, ist eine Verbindung mit dieser Adresse nicht möglich. Im darunter liegenden Kollektor Fenster ist der Verbindungswunsch jedoch als gelungen eingestuft und abgespeichert worden. Auch der angeblich gelungene Verbindungswunsch mit der Seite *www.focus.de* entspricht nicht der Wahrheit. Beide Messungen wurden erzeugt, indem die entsprechende URL im Browser eingegeben und anschließend das Fenster manuell mit der Taste F5 refreshed wurde.

Desweiteren werden Funktionsknöpfe der Anwendung nicht 100 % korrekt überwacht. Hierzu ein weiteres Beispiel.Abb. 3.3 zeigt einen leeren Browser und einen Kollektor, zu erkennen an einem andauernden "*Connect*" Status. Betrachtet man den Browser etwas genauer, erkennt man, dass der Verbindungswunsch mit *www.detesystem.de* bereits abgebrochen wurde, zu erkennen am Fehlen der Fortschrittsanzeige in der untersten Zeile des Browserfensters, sowie am leeren Fensterinhalt. Der Abbruch erfolgte dabei über den roten Abbruch Button in der Funktionsleiste. Wie Abb. 3.3

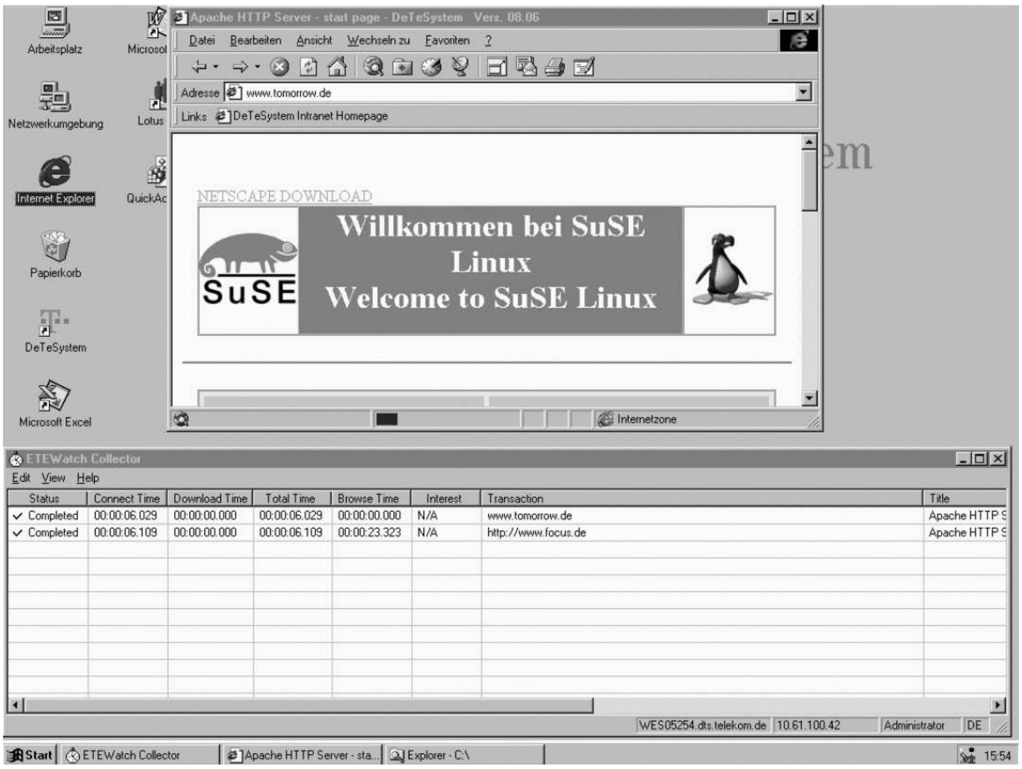


Abbildung 3.2: Transaktion fälschlicherweise COMPLETE

zeigt, hat der Kollektor diesen Abbruch nicht erkannt und erwartet eine Verbindungsrückmeldung, die jedoch niemals stattfinden wird.

Würde anschließend eine Eingabe einer erreichbaren Seite erfolgen, z.B. *www.testbed.de*, würde der Kollektor den erfolgreichen Seitenaufbau dieser neuen Seite, als erfolgreichen Verbindungsaufbau zur Seite *www.detesystem.de* zuordnen. Obwohl nun z.B. die Seite *www.testbed.de* vollständig im Browser zu sehen wäre, enthält das Kollektor Protokoll einen Eintrag über die erfolgreiche Verbindung zur Seite *www.detesystem.de*. Gleiches gilt bei FTP Downloads über den Browser. Wird eine FTP Anfrage mittels des Abbruch Buttons abgebrochen, bevor ein Download überhaupt stattfinden konnte, erkennt der Kollektor diesen Abbruch nicht und verursacht wie im oberen Fall einen unkorrekten Eintrag.

Wird im Gegensatz dazu ein Verbindungsaufbauwunsch mit ESC abgebrochen, protokolliert der Kollektor korrekt mit und stuft den jeweiligen '*Connect*' Aufruf als Abbruch ein. Grundsätzlich ist zu bemerken, dass die hier aufgezeigten Fehler keineswegs von einer derart gravierenden Natur sind, dass sie nicht durch bessere Aufmerksamkeit in der Programmierung der Überwachungsmodule eliminiert werden könnten. Zudem wird die Behandlung von Tastatureingaben richtig ausgewertet.

Ausgehend von den Ergebnissen des Browsertests, ist lediglich eine eingeschränkte Verlässlichkeit auf Messdaten in dieser Version vorhanden. Aufgrund der Mängel bei Abbruch Anforderungen und der damit falschen Interpretation des Kollektors, kann keine akkurate Messung gewährleistet werden. Es sei nochmals gesagt, dass es sich bei diesem Test nur um eine Überwachung eines Webbrowsers gehandelt hat. Aussagen über die Überwachung von Lotus Notes oder SAP/R3 werden nicht gemacht.

Positiv ist jedoch anzumerken, dass sich das Produkt absolut einfach in Handhabung und Konfi-

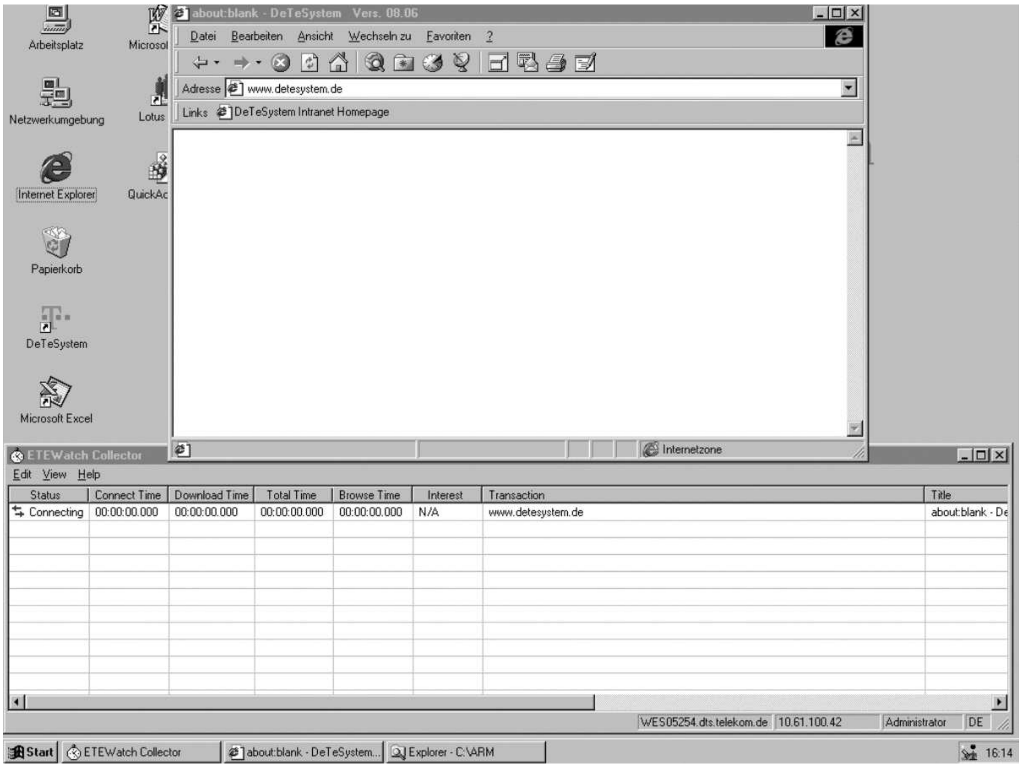


Abbildung 3.3: Transaktion fälschlicherweise ANDAUERND

guration darstellt. Weder großer technischer Aufwand bei der Installation, noch große Ansprüche im Bezug auf Ressourcen sind zu erkennen. Dafür ist einfache und benutzerfreundliche Interaktion mit der Software gewährleistet. Desweiteren werden Module für alle gängigen Büroanwendungen, wie z.B. Lotus Notes oder SAP/3 angeboten.

3.1.5 Ressourcenverhalten

Der Einsatz von Kollektoren geschieht durch Installation und Initialisierung auf jedem zu messenden End-System. Um den Betrieb das Endsystem so wenig wie möglich zu beeinflussen, sollen Kollektoren sich weitestgehend zurückhaltend im Bezug auf Systemressourcen verhalten. Candles ETEWATCH Kollektor verhält sich in diesem Punkt vorbildlich. Es werden nur ca. 3 MB Speicher belegt und eine CPU Belastung ist so gut wie nicht feststellbar. Geht man von heutigen Rechnern aus, kann die Gesamtbelastung des Systems durch den Kollektor als unbedeutend eingestuft werden und gewährleistet so einen ungestörten Betrieb.

3.1.6 Dokumentation

Beispielhaft ist die komplette Online Dokumentation der einzelnen Module von Kollektor, Client, Manager bis hin zum Reporter. Zu jedem Menüpunkt gibt es einen Verweis auf die entsprechenden Hilfeseiten, in der jeder Einstellungsparameter genauestens beschrieben wird.

3.1.7 Support

Der Support bezieht sich auf jeden Aspekt für die Inbetriebnahme dieser Software. Generell wird hauptsächlich auf die Bereitstellung und Programmierung neuer Module hingewiesen. Sollen kundenspezifische Applikationen überwacht werden, kann die Firma Candle innerhalb weniger Tage oder sogar Stunden, abhängig von der Komplexität der Anwendung, neue Verhaltensmodule dafür erstellen. Verhaltensmodule beschreiben die Oberfläche und die Auswirkungen von verschiedenen Aktionen. D.h. es werden Oberflächeneigenschaften oder -objekte deklariert, die einen Start oder Endpunkt in der Transaktion markieren. Damit ist es möglich den Kollektor an jede Art von Anwendung anzupassen.

3.1.8 Kosten

Die Kosten für den Einsatz von *Etewatch* berechnen sich wie folgt. *Etewatch* for Networked Applications Starter Kit, verfügbar für SAP, Lotus Notes & Web Browsers, incl. 1 Behaviour Module und bis zu 100 Workstations: DM 45.500

Weitere 150 Workstations: jede Workstation DM 364,–
Weitere 250 Workstations: jede Workstation DM 182,–.
Weitere 500 Workstations: jede Workstation DM 91,–.

3.1.9 Einsatz

Folgende Firmen benutzen Etewatch momentan zur Antwortzeitüberwachung: Debis, Zürich Versicherung, Ostdeutscher Rundfunk, RWE Energie, Siemens, Wella, Shell, EDS, France Telcome.

3.1.10 Weiterentwicklung

Wie schon erwähnt ist dieser Ansatz gänzlich ungeeignet um das Antwortzeitverhalten bei anonymen E-Commerce Nutzern zu messen. Candle hat aber auch hier eine Methode entwickelt um diesem Anspruch gerecht zu werden. '*eBA*' (*E-Business Assurance*) heißt die Lösung um auch von anonymen '*Internetnutzern*' Messungen zu erhalten. Tatsache ist, Antwortzeitmessungen auf Applikationsebene können nur am Benutzer PC selbst gemessen werden und es gibt keine Möglichkeit von einem anderen PC aus, diese Messung durchzuführen solange kein Agent auf der zu messenden Maschine installiert ist. eBA beschränkt sich auf Messungen des Webbrowser und des Seitenaufbaus. Es benutzt ein Java Applet, welches beim Öffnen der Seite mitschickt wird und als Messagent fungiert. Es werden Zeiten für den Seitenaufbau ermittelt und diese zurück an den Service Provider geschickt. Nebenbei verschickt das Applet noch eine Reihe weiterer, für den Webanbieter nützlicher Informationen, wie verbrachte Zeit auf den einzelnen Seiten oder die Reihenfolge der Seitenaufrufe. Informationen dieser Art ermöglichen dem Seitenanbieter Webauftritte effizienter und kundenfreundlicher zu gestalten. Auch im Hinblick auf den Datenschutz wird eBA den Anforderungen gerecht. Es werden keine persönlichen Daten, wie IP oder Namen aufgezeichnet, sondern nur das Browserverhalten und die angeforderten Seiten. Ein äußerst vielversprechendes Produkt, dass einen innovativen Ansatz bietet um E-Commerce noch besser zu machen.

3.2 Firehunter 3.02 von Agilent/HP

Firehunter ist ein System, das sich auf Agenten stützt, die aktive sowie passive Messungen von Service Qualität durchführen.

Es besteht aus einem Diagnostic Measurement Server, kurz DMS genannt und einem oder mehreren Agenten. Der DMS ist zentrale Anlaufstelle für die Verwaltung, Konfiguration und Visualisierung der gesammelten Daten. Die Agenten können aktive wie auch passive Messungen durchführen und ihre Ergebnisse an den DMS Server weiterleiten. Aktive Messungen erzeugen Anfragen an Netze, Server oder Service Applikationen um Aussagen über Verfügbarkeit, Auslastung, Antwortzeitverhalten abzuschätzen. Z.B. um die Verfügbarkeit und die Performance eines Webservers zu testen, kann eine aktive Messung ein HTTP GET Kommando für eine Webseite initiieren und den Status sowie die Gesamtdauer des Downloads überwachen.

Die passiven Messungen hingegen beschränken sich darauf serverseitigen Verbindungs- auf bzw. -abbau zu registrieren und zu analysieren.

Firehunter kann unter das Konzept der 'Simulierten Anfragen' eingeordnet werden, da es sich hierbei um ein dediziertes System handelt, dass sich durch simulierte Serviceanfragen, wie Web, Mail, FTP oder DNS ein Bild über das Antwortzeitverhalten der angesprochenen Services macht. Abb. 3.4, zeigt den schematischen Einsatz einer Firehunter Überwachung mittels DMS. In einem Firmennetz, in der Graphik rechts zu sehen, wird ein DMS Server installiert, der benutzte Services bei einem Service Provider (unter Firmennetz) und/oder seines ISP aktiv überwacht. Zusätzlich werden passive Agenten auf der Seite des ISP eingesetzt um einen besseren Überblick über die Auslastung des ISP zu bekommen. Schlechte Antwortzeiten, z.B. durch überlasteten DNS Server des ISP, können frühzeitig erkannt und behoben werden.

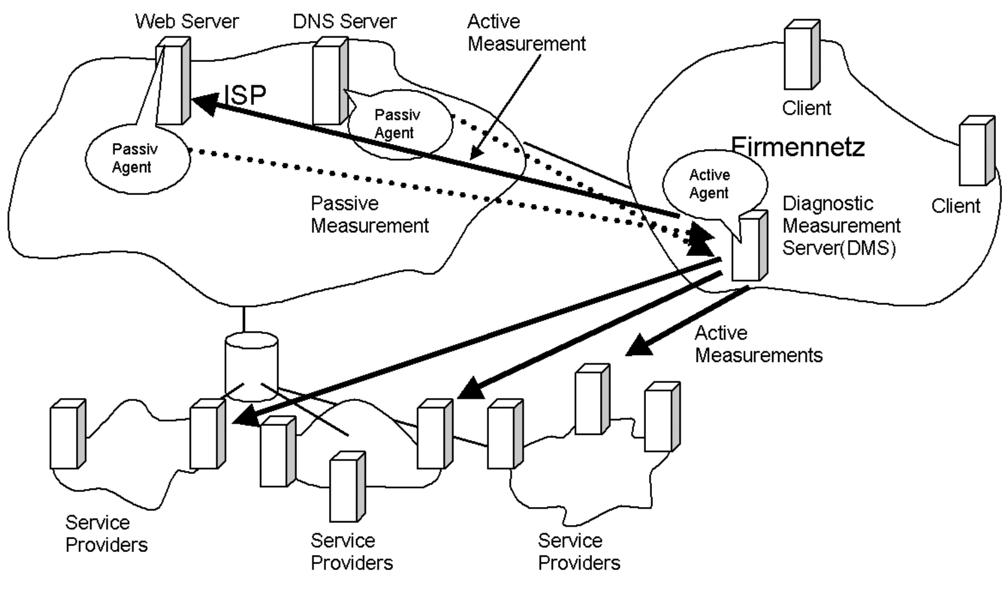


Abbildung 3.4: Beispiel eines DMS Aufbaus (nach [Agi])

3.2.1 Systemvoraussetzung

Im folgenden sind die Systemanforderungen an eine Firehunter Umgebung aufgelistet.

DMS Diagnostic Measurement Server	
OS	NT 4.0, Server oder Workstation, Service Pack 5 (Intel) Solaris 2.6, 2.7 (SPARC) HP-UX 11.00
CPU	NT: Pentium-II 300 Mhz (minimum), 500 Mhz od. schneller (empf.) Solaris: 333 Mhz Ultra-5 (minimum) HP-UX: 240 Mhz C240 (minimum), 400MHz C3000 (empfohlen)
Disk	Installation: 120 MB Speicheranforderungen bei Datenbankbenutzung : - 25 KB/Messung/Tag (ca.) bei 5 min. Messungsfrequenz - Service Model mit 150 Messungen: 1.5 GB jährl. (ca.)
Speicher	Minimum: 192 MB RAM empfohlen: 256 MB RAM
Virtueller Speicher	ungefähr 256 MB

Remote GUI	
OS	siehe DMS Systemvoraussetzungen
Disk	Installationsanforderungen variieren von OS zu OS, ungefähr 30 MB.
Speicher	Speicheranforderungen variieren von OS zu OS. Typische Speicheranforderungen für das Remote GUI, sind substantielle weniger als für das Firehunter DMS System. 32 MB sind Minimum; Bei komplexen Graphen oder großen Service Modellen könnte mehr benötigt werden .
Virtual Memory	ungefähr 128MB

Remote Measurement Agent	
OS	All listed in DMS System Requirements, plus: - RedHat Linux 6.1, with glibc version 2.1.2 or greater - HP-UX 10.20 - FreeBSD 3.1,3.2,3.3 - Solaris 2.5.1 (SPARC) - Solaris x86 2.6,2.7
Disk	Installationsanforderungen variieren von OS zu OS, ungefähr 30 MB.
Memory Usage	Speicheranforderungen variieren von OS zu OS. Typische Speicheranforderungen für einen Remote Agent sind geringer als für das Firehunter DMS System . 64 MB sind Minimum;Bei großen Service Modellen könnte mehr benötigt werden.
Virtual Memory	ca. 256 MB

3.2.2 Komponenten von Firehunter 3.02

3.2.2.1 DMS - Diagnostic Measurement Server

Die Firehunter Architektur besteht aus drei Objekten, den Agenten, dem Auswertungstool und dem Server, hier DMS (Diagnostic Measurement Server) genannt. Der DMS vereint Datenbankaufgaben mit Reportfunktionalität und stellt damit die zentrale Verwaltungsschnittstelle von Firehunter dar. Einerseits ist der DMS Sammelpunkt für Messdaten der Agenten, die mittels FTP an ihn übertragen werden um dort korelliert und ausgewertet zu werden. Zusätzlich gewährleistet der DMS einen allgemeinen Zugang zu Messdaten, indem Ergebnisse z.B. auf HTML Basis visualisiert und diese anschließend über einen installierten Webserver zugänglich gemacht werden. Die Konfiguration der Agenten geschieht über eine eigenständige Admin Console. Diese Eingabeschnittstelle stellt eine Verbindung mit dem DMS her, der die Einstellungen schließlich an die einzelnen Agenten weiterleitet. Andererseits können mit dem DMS Schwellwerte für bestimmte

Dienste festgelegt und schließlich bei Überschreitung geeignete Ausnahmebehandlungen erfolgen. Solche Behandlungen beinhalten die Benachrichtigung einer verantwortlichen Stelle oder Person. Diese Benachrichtigung kann z.B. über Pager, E-Mail oder Webseite geschehen. Diese Aktionen lassen sich individuell zusammensetzen und sind flexibel einsetzbar.

3.2.2.2 Firehunter Agenten

Agenten werden in zwei Kategorien unterteilt und zwar in Aktive und Passive. Aktive Agenten erzeugen Netzverkehr indem sie Benutzerähnliche Anfragen, wie z.B. Webseitenaufruf oder E-Mail Abholung, an Services stellen und diese messen. Die aktiven Firehunter Agenten sind dabei in Einzelkomponenten aufgeteilt, die jeweils getrennt ihre Messungen durchführen. Eine Webseitenanfrage wird z.B. in die Teile DNS Lookup Time, TCP Connection Time, Server Response Time und schließlich Data Transfer Time zerlegt. Sowohl die Einschätzung der Service Qualität, wie auch Problemerkennung ist damit möglich. Passive Agenten erlauben zusätzliche Diagnose der Verbindungen von und zu Servern und Diensten. Wie der Name schon sagt, handelt es sich hierbei um eine nicht Verkehrsgenerierende Messmethode, sondern um eine passive Überwachung von Netzressourcen. Der Einsatz dieser Agenten umfasst hauptsächlich die Aufzeichnung von Verbindungsstatistika, wie momentane Verbindungen oder Verbindungswünsche. Das sind z.B. unter anderem Accepted Connections per Sec, Requested Connections per Sec oder Established Connections per Sec. Zusatzinformationen dieser Art können mit aktiv gemessenen Zeiten kombiniert werden und erlauben so detailliertere Einsichten in die Gesamttransaktion.

3.2.2.3 Firehunter GUI Application

Die Gui Application von Firehunter ist ein Report und Visualisierungs Tool, das gesammelte Daten veranschaulicht. Dabei wird eine Verbindung mit einem DMS Server hergestellt um auf die gesammelten Daten zugreifen zu können. Der Application GUI bietet dabei die Möglichkeit einzelne oder mehrer Messungen graphisch anzuzeigen und Ausnahmebehandlungen bei Schwellwertüberschreitungen zu konfigurieren. Eine Benachrichtigung über kritische Zeiten, kann je nach Bedarf über E-Mail, SNMP Traps, Pager oder Webseiten erfolgen. Die Einstellungen über Art und Umfang legt man im Application GUI fest. Der DMS übernimmt anschließend die Aufgabe diese Nachrichten zu erzeugen und zu verteilen, vorrausgesetzt er ist an die entsprechenden Dienste angeschlossen.

Abb. 3.5, erläutert nochmals grob die Struktur und die Bestandteile eines Firehunter Systems.

3.2.2.4 Leistung

Die Installation ist problemlos und unkompliziert. Man hat die Möglichkeit einzelne Module oder das gesamte Paket von Firehunter auf dem PC zu installieren. Ein lauffähiges System besteht jedoch immer aus DMS Server, einem oder mehreren Agenten und der GUI Application zur Auswertung. Zum Test wurde das gesamte Paket auf einer Maschine installiert und erlaubte so auf Messdaten des Agenten direkt zuzugreifen. Abb. 3.6 zeigt die Admin Console und ihre verfügbaren Einstellungen. Diese Console ist als Eingabeschnittstelle zu betrachten und erlaubt eine benutzerfreundliche Konfiguration der Agenten. Für Standardanwendungen sind bereits zahlreiche, vorkonfigurierte Templates (links im Bild) vorhanden, die einfach übernommen werden können. Diese Templates benötigen nur sehr wenige Informationen, um eine allgemeine Überwachung zu starten. Meistens benötigt man lediglich IP oder Hostnamen einer Maschine und den Dienst der überwacht werden soll (Newsserver, Webserver , Mail,...).

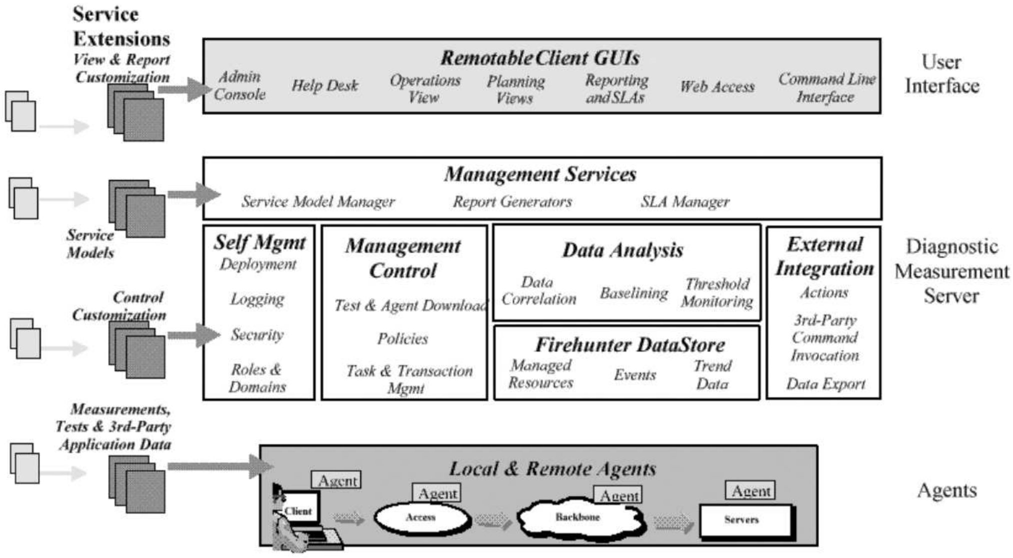


Abbildung 3.5: Firehunter Architektur (aus [Agi])

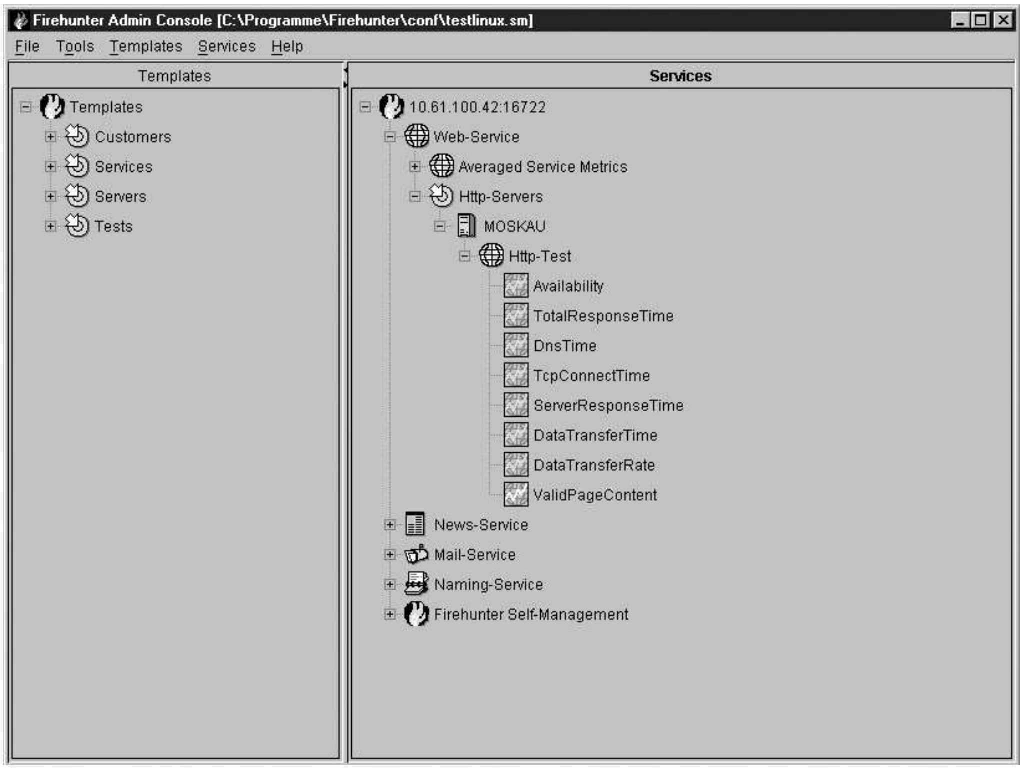


Abbildung 3.6: Admin Console von Firehunter zur Konfiguration der Agenten

Eine genauere Spezifizierung der Überwachungsparameter, wie z.B. URL einer WebSeite, FTP Zugangsaccount, FTP File, ist unter den Eigenschaften des ausgewählten Überwachungsservices zu finden. Sehr gelungen ist die Strategie zur Überwachung von *Service Level Agreements* (SLA) für einzelne Kunden, welche einen eigenen Menüpunkt unter den Test Templates der *Admin Console* besitzen. Dies erlaubt, auf Garantien an einzelnen Kunden getrennt einzugehen, zu managen und zu berichten. Dabei werden die Ergebnisse einer Überwachung unter einem Kundenprofil zusammengefasst und anhand der ausgehandelten Qualitätsmerkmale bewertet.

Entsprechende Schwellwertüberschreitungen werden dabei gleich in Kosten umgewandelt und in eine Wochen- oder Monatsbilanz gebracht. Dazu ein kleines Beispiel. Ein Unternehmen mietet sich bei einem beliebigen Provider einen Webserver um ihre Angebote online zu präsentieren. Der Provider garantiert für 100% Verfügbarkeit und schnellen Zugang zu den Seiten. Zusätzlich wird vereinbart bei Nichteinhaltung der Zusicherungen (SLA's) die Mietkosten entsprechend zu senken. D.h. pro Überschreitung der Qualitätsvereinbarungen (z.B. Download einer Webseite hat länger als 20 Sekunden gedauert) werden die Kosten für das Unternehmen um einen vereinbarten Betrag verringert.

Weiterhin ermöglicht die SLA Funktionalität von *Firehunter* Berichte produktiv zu trennen. Während der Kunde primär nur an einer Zusammenfassung und Kostenbilanz interessiert ist, möchte der Provider natürlich auch die einzelnen Nichteinhaltungen genauer studieren können um eventuell weitere zu verhindern. Beides ist mit *Firehunter* möglich, da SLA-Reports unterschiedlicher Tiefe und Perspektive angefertigt werden können.

Darüber hinaus gestaltet sich der Zugriff auf diese Berichte schnell und unkompliziert. *Firehunter* generiert Reports bereits im HTML Format und bietet sie gleichzeitig durch seinen eingebauten Webserver online an. Abb. 3.7 zeigt eines der Firehunter SLA Templates. Die Service Überwachung umfasst in diesem Beispiel einen Webservice (WebGoldSLA) mit einer sehr hohen Qualitätsanforderung (Web*GOLD*SLA), wie unter den Punkten Required Compliance und Web Availability zu erkennen ist. Required Compliance von 98.9% bedeutet in diesem Fall eine fast vollständige Einhaltung der Vereinbarung, ansonsten kann eine Verminderung der Kosten die Folge sein. Web Availability Violation 100% gibt an, dass jede Grenzwertüberschreitung zur Nichteinhaltung gezählt wird. Und Web Weekday/Weekend Response Time Violation von 3.0, erlaubt eine dreimalige Überschreitung der Antwortzeitgrenzwerte. Jede weitere wird als Verletzung der Qualitätszusicherungen angesehen.

Auf der Basis dieser Angaben, kann ein zusammenfassender Report nach durchgeführter Überwachung aufgestellt werden. Abb. 3.8 zeigt einen möglichen Beispielreport, dessen Detaillierungsgrad natürlich variabel einstellbar ist.

Nun noch zu einer kleinen Beispielüberwachung eines Web Servers. Auf Abb. 3.6 konnte man bereits eine mögliche Konfiguration der Überwachungsparameter sehen. Diese Einstellungen entstammen eines vorgefertigten Templates zur Webüberwachung und sollen auch die Grundlage des folgenden Tests sein. Dazu gehören Availability, TotalResponseTime, DNSTime, TCPConnectTime, ServerResponseTime, DataTrasferTime und DataTransferRate eines Webservers. Der Agent schickt nun simulierte Anfragen an den Webserver im Intervall von 5 Minuten und überträgt die Ergebnisse sofort nach Erhalt an den DMS Server. Dieser speichert die Daten und überprüft sie auf Ausnahmebehandlungen. Mit dem Application GUI können schließlich die Daten angezeigt werden, was man in Abb. 3.9 sehen kann.

Im linken unteren Fenster werden die definierten Messungen aufgelistet. Im rechten Fenster können ihre Ergebnisse anhand von Graphen visualisiert werden. Auf Abb. 3.9 sieht man z.B. die Auswertung der TotalResponseTime des http-Test der Maschine '*MOSKAU*'. Die Waagerechten farbigen Streifen im rechten Teil des Bildes sind festgelegte Thresholds für verschiedene Ausnahmebehand-

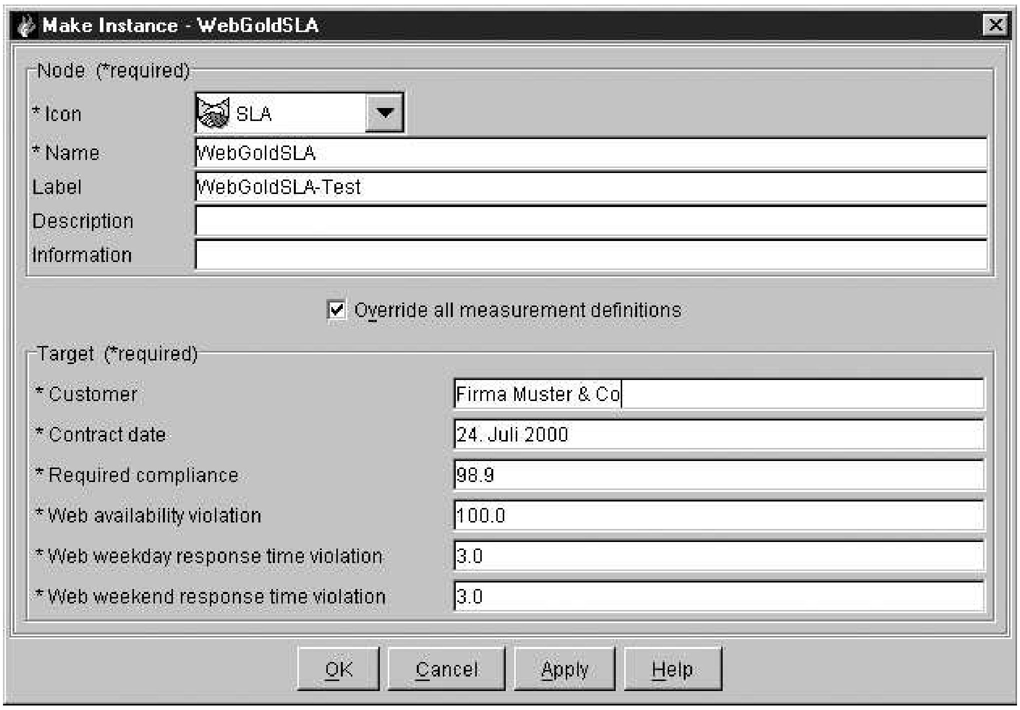
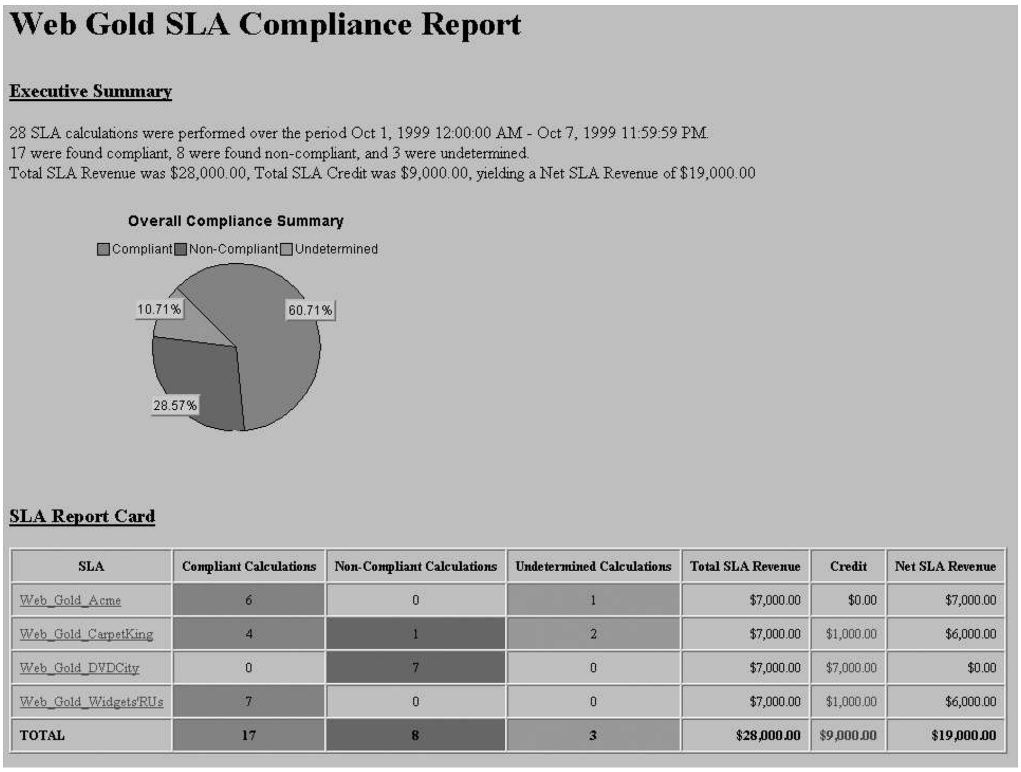


Abbildung 3.7: SLA Konfigurationsmenü



SLA	Compliant Calculations	Non-Compliant Calculations	Undetermined Calculations	Total SLA Revenue	Credit	Net SLA Revenue
Web_Gold_Acme	6	0	1	$7,000.00	$0.00	$7,000.00
Web_Gold_CarpetKing	4	1	2	$7,000.00	$1,000.00	$6,000.00
Web_Gold_DVDCity	0	7	0	$7,000.00	$7,000.00	$0.00
Web_Gold_Widgets'RUs	7	0	0	$7,000.00	$1,000.00	$6,000.00
TOTAL	**17**	**8**	**3**	**$28,000.00**	$9,000.00	**$19,000.00**

Abbildung 3.8: SLA Beispiel Report

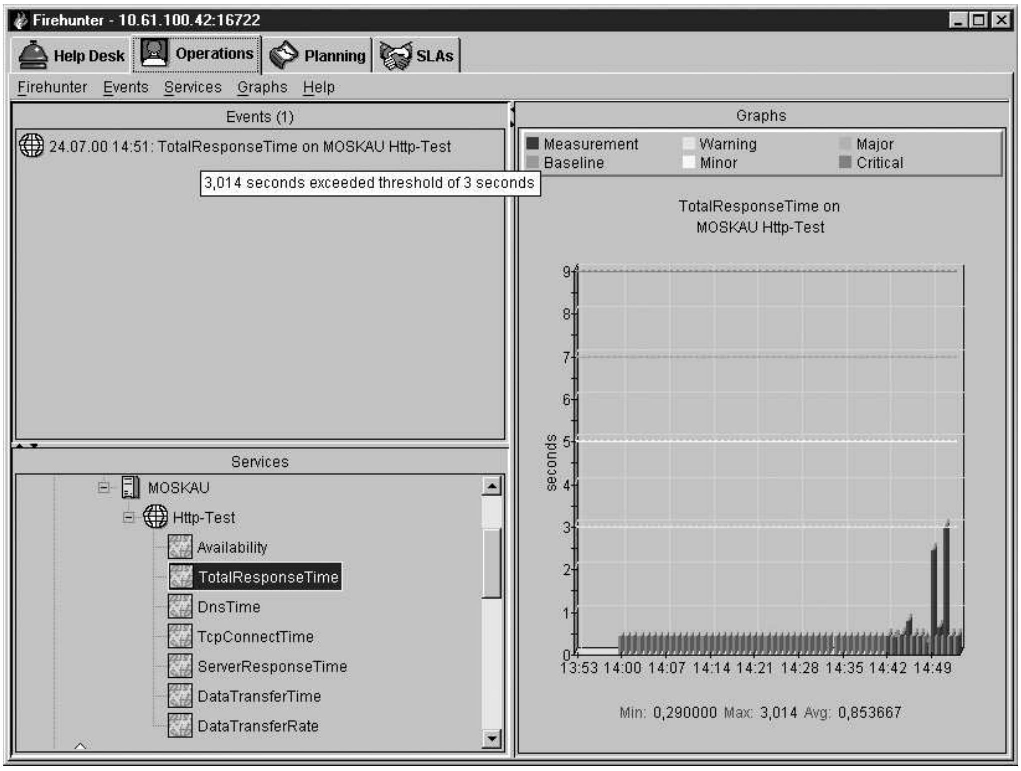


Abbildung 3.9: Application GUI

lungen. In unserem Beispiel wurde der unterste Schwellwert von 3 Sekunden einmal um ca.14.50 Uhr überschritten, was zu einem Event geführt hat. Events sind Ausnahmebehandlungen, die durch Überschreiten von Grenzwerten ausgelöst werden. Dieses Event sieht man im linken oberen Teil der Abb. 3.9. Dort ist noch mal der genaue Zeitpunkt, der Service und der Grad der Schwellwertverletzung zusammengefasst. Zusätzlich zur Benachrichtigung über das Auswertungsportal Application GUI, kann diese Nachricht natürlich auch über E-Mail oder Pager weitergeleitet werden.

3.2.3 Kosten

Grundsätzlich erwirbt man ein Firehunter-Grundpaket, welches dann mit sogenannten 'Plug-In Modulen' aufgerüstet wird. Beispiele dafür ist das *Network Service Model*-Modul und das *E-Commerce*-Modul. Das Firehunter-Grundpaket richtet sich in seinen Kosten nach der Anzahl der möglichen Messungen. Damit ergibt sich eine Skalierbarkeit im Bereich von 30.000 bis 4 Millionen Euro.

3.2.4 Einsatz

Firehunter befindet sich bereits im Einsatz unter anderem bei folgenden Firmen:

- Network Monitoring: BMW AG München BMW nutzt Firehunter, um das firmeneigene WAN zu überwachen, da die einzelnen Abteilungen mit der IT-Abteilung konsequent Service Level Agreements über die Dienstgüte vereinbaren.

- E-Commerce: HP Shopping HP Shopping nutzt Firehunter, um die Leistungsfähigkeit der eigenen E-Commerce Shopping Mall zu überwachen. Hier kommt Firehunters E-Commerce Modul zum Einsatz.
- Antwortzeitüberwachung: Qwest Communications/USA Qwest benutzt Firehunter, um die Leistungsfähigkeit des eigenen amerikaweiten Netzes zu überwachen. Antwortzeiten stehen hier natürlich im Mittelpunkt der Messungen.

3.2.5 Zusammenfassung:

Als Vertreter des Konzepts der Simulierten Anfragen, macht dieses Produkt einen überaus professionellen und kompetenten Eindruck. Leider wird dieses positive Bild durch seine hohen Systemanforderungen etwas getrübt. Bei minimal Anforderungen von 192 MB Speicher für einen DMS Server und minimal 64 MB Speicher für jedes Agentensystem, lassen Kosten für Hardware nicht unbedingt nebensächlich erscheinen. Noch einige Kommentare zu Dokumentation, Datenbankanbindung und Verfügbarkeit. Das Hilfe System ist außerordentlich gut implementiert und enthält zu jedem Menüpunkt ausführliche Informationen und erleichtern so ein schnelles Zurechtfinden mit der Software. Die Datenbankanbindung kann über ODBC Treiber realisiert werden, was den Datenaustausch mit anderen Datenbanken ermöglicht. Ein ausschlaggebender Vorteil dieses Produkts ist die Plattformübergreifende Erhältlichkeit. Das Firehunter Paket ist sowohl für Solaris, WinNT und UNIX Plattformen erhältlich.

3.3 INFRA-XS oder auch HCW (Hitnet Communications Watch) von Geyer & Weinig

Infra-XS als Client-Server basiertes realtime Monitoring Tool, kann als gemeinsamer Vertreter zweier Konzepte auftreten. Das Prinzip dieses Tools ist es, sowohl mittels Beispielanfragen Messungen durchzuführen, wie auch auf der Netzwerkebene ein gewisses Maß an Analysen vorzunehmen. Ergebnisse resultieren dabei aus Kombination beider Messverfahren (WinXS und FrameXS). Abb. 3.10 beschreibt den Aufbau des *Infra-XS Systems*. Er besteht aus 3 Hauptkomponenten, den *XS-Agenten*, den *XS-Servern* und den *XS-Clients* aus ([Doc]).

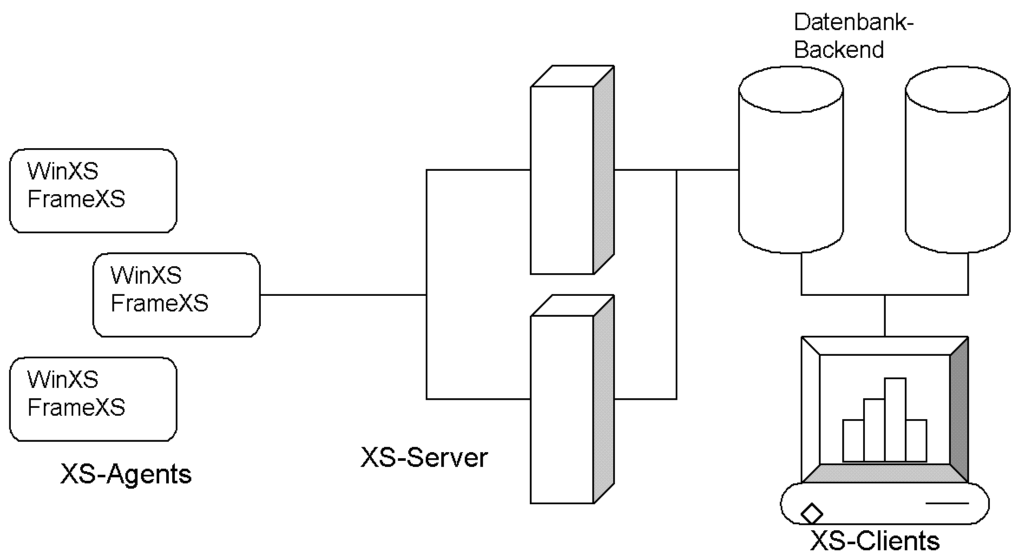


Abbildung 3.10: Infra-XS Komponenten Aufbau nach ([Doc])

3.3.1 Systemvoraussetzungen

Folgende Tabelle gibt über die Systemvorraussetzungen aller Komponenten des Produktes *Infra-XS* Auskunft.

Infra-XS alle Komonenten	
OS 16-Bit Version	MS-DOS ab Version 5.0 Microsoft Windows WfW 3.11 Od. Microsoft Windows 3.1 mit Widows Socket API Version 1.1 kompatiblen Stack
OS 32-Bit version	Microsoft Windows 95 Windows NT 3.51 oder 4.0
CPU	Min. Intel 486DX2/66 empfohlen Intel Pentium/166
RAM	Min. 8MB, empfohlen 32 MB
Disk	Ca. 20 MB für Agenten Ca. 40 MB für Server Systeme

3.3.2 Funktionsweise

Zunächst zur technischen Realisierung der Antwortzeitüberwachung dieses Tools. Das Monitoring Tool *Infra-XS* benutzt eine Kombination aus Anfragensimulation und Netzanalyse um Antwortzeiten zu überwachen. Zwei Komponenten spielen dabei eine wichtige Rolle, WinXS und FrameXS. Mittels WinXS Notifier kann eine Transaktion aufgezeichnet werden, um sie später ohne Benutzereingriffe automatisiert ablaufen zu lassen. Dabei werden z.B. Mausposition, Tastatureingaben, Mausklicks abgespeichert und ergeben ein festes Eingabeprofil für die Anwendung. Die Automatisierungs-Komponente (WinXS Scriptprozessor) erlaubt die Ausführung solcher Scripts, die eine graphische Anwendung so ablaufen lassen, wie es dem Benutzerverhalten bei Referenztransaktionen entspricht. Die Eingabebereitschaft bestimmter transaktionsrelevanter Fenster oder das Anzeigen von Transaktionsergebnissen in Tabellen oder Fenstern führen zur Aufzeichnung relevanter Messwerte. Damit zwischen Anwendungs- und Service bezogenen Messdaten unterschieden werden kann, müssen entsprechende Counter und Servicepunkte gesetzt werden. Diese Servicedefinitionen ermöglichen schließlich die Zuordnung eines automatisierten Ablaufs zu einer Referenztransaktion und die Aufteilung in PC-, netz- und hostbasierte Anteile. Z.B. könnte eine Mailüberwachung vorsehen, dass ein Account geöffnet wird, alle Mails heruntergeladen werden und anschließend das Programm beendet wird. Würde man diese Transaktion in seiner Gesamtheit betrachten, würde sowohl Start des Mailprogramms, Auswahl der Mails und das Schließen des Mailprogramms zur Transaktionszeit dazugezählt. Relevant ist dabei jedoch nur die Zeit in der die Mails heruntergeladen werden. Um das zu realisieren müssen sog. Servicedefinitionen in das Script miteingebaut werden. D.h. Der Start des Downloads, was z.B. als Folge ein neues Fenster öffnet, könnte als Startpunkt der Mailtransaktion angesehen werden, wobei ein 'Complete' String in einem Fensterabschnitt als Ende zu werten wäre. Diese Punkte müssen innerhalb des ablaufenden Scripts genauestens definiert werden, damit eine Differenzierung zwischen PC und Netz Anteil überhaupt geschehen kann. Infra-XS geht aber noch einen Schritt weiter. Und zwar wird, um dieses Transaktionsfenster in weitere Einzelteile zerlegen zu können, zusätzlich zu WinXS die Komponente FrameXS benutzt. FrameXS misst auf Netzwerkebene und überwacht innerhalb des von WinXS vorgegebenen Zeitfensters TCP spezifische Aufrufe und Übertragungen. FrameXS erreicht damit eine Aufteilung der Gesamttransaktion in ihre Einzeltransaktionen, wie DNS Lookup Time, Data Transfer Time usw. Diese erhöhte Detaillierungsstufe der Transaktionszeiten macht nicht nur gemessene Transaktionszeiten genauer, sondern erleichtert dadurch auch Fehlerdiagnosen.

3.3.3 Infra-XS Komponenten

3.3.3.1 XS-Server

Der *XS-Server* ist die zentrale Verarbeitungskomponente im *Infra-XS* Umfeld. Die von den Agenten erhobenen Daten werden hier gesammelt, aufbereitet und mit einer Datenbank abgeglichen. Replikationsmechanismen bieten den gesicherten Datentransfer zu einem *XS-Server* der nächsthöheren Ebene in einem kaskadierenden Serverumfeld. Dieser Serveraufbau kann eingesetzt werden, falls eine verteilte Überwachungsverantwortlichkeit vorliegt, beispielsweise applikationsorientiert, regional oder gesamt.

3.3.3.2 XS-Clients

Die *XS-Clients* übernehmen die Aufgabe des Designs von Auswertungen und Reports und das Monitoring von Realtime-Daten. Dabei kommunizieren sie entweder direkt mit den entsprechenden XS-Servern oder mit einem oder mehreren Datenbanksystemen. Die Kommunikation zu den

Datenbanksystemen erfolgt über entsprechende ODBC- oder native Treiber, z.B. für INFORMIX, ORACLE, SYBASE, INTERBASE, DB2 etc.

3.3.3.3 XS-Agents

Die *XS-Agenten* messen, generieren und verteilen Applikations- und Netzwerkrelevante Daten von End-to-End Kommunikationsbeziehungen. Die Agenten laufen bedienerlos und vollständig automatisiert auf Referenz-PC-Systemen. Mittels ScriptProzessor werden erforderliche Tastatureingaben oder Mausbewegungen einer relevanten Anwendung gesteuert und erzeugen ein Benutzerähnliches Eingabeverhalten. Speziell eingefügte Markierungen und Überwachung von Servicedefinitionen (z.B. Login Fenster schließt sich bei gelungener Anmeldung) gestatten Anwendungsbezogene Messungen.

3.3.4 Leistung

Infra-XS ist ein überaus komplexes und umfangreiches Monitoring Tool, dass seinen Einsatz dementsprechend schwer gestaltet. Bereits bei der Installation stößt man auf Schwierigkeiten, die nur mit Hilfe von fachkundiger Auskunft überwunden werden können. Ein betriebsbereites System besteht aus den drei Komponenten *XS-Agents*, *XS-Server* und *XS-Clients*. Zuerst sollte ein *XS-Server* installiert werden um den Agenten einen Anlaufpunkt zu geben. Er benötigt dazu eine Datenbank, für die Speicherung der Messdaten, sowie eine Übertragungskomponente (FTP, MAIL oder SNMP usw.), damit er Daten der Agenten empfangen kann. Alleine das aufsetzen der Datenbank verlangt ein hohes Maß an Wissen über Betrieb und Funktionsweise des *Infa-XS Servers*, dass einem die Handbücher leider nur mühsam vermitteln können. Wie auf Abb. 3.11 zu sehen, erlaubt das Konfigurationsprogramm vielfältige Einstellungen an jeder Komponente vorzunehmen. Gerade wegen dieser unglaublichen Konfigurierbarkeit, ist es sehr schwierig sich zurechtzufinden.

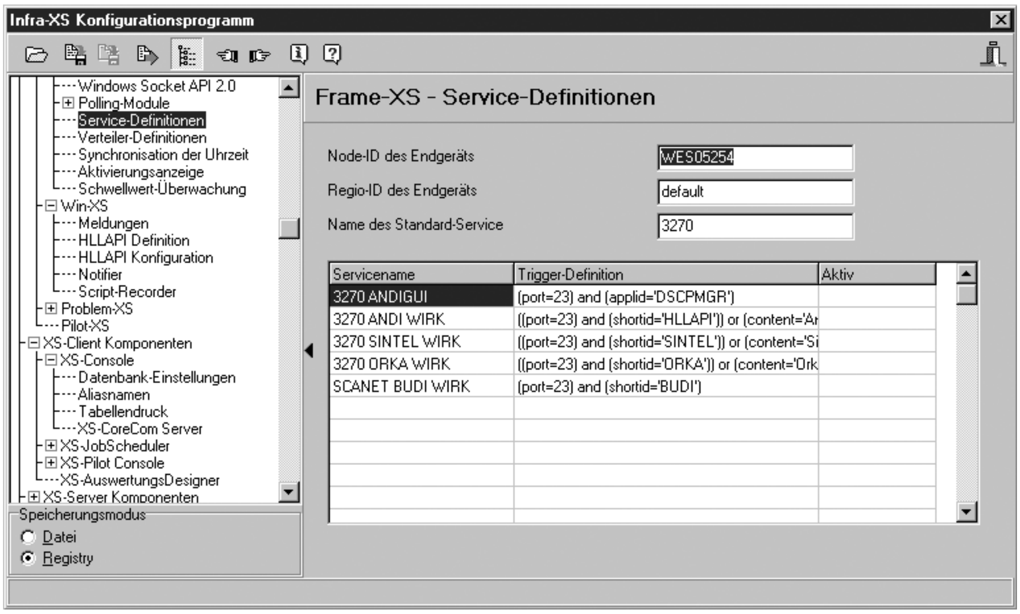
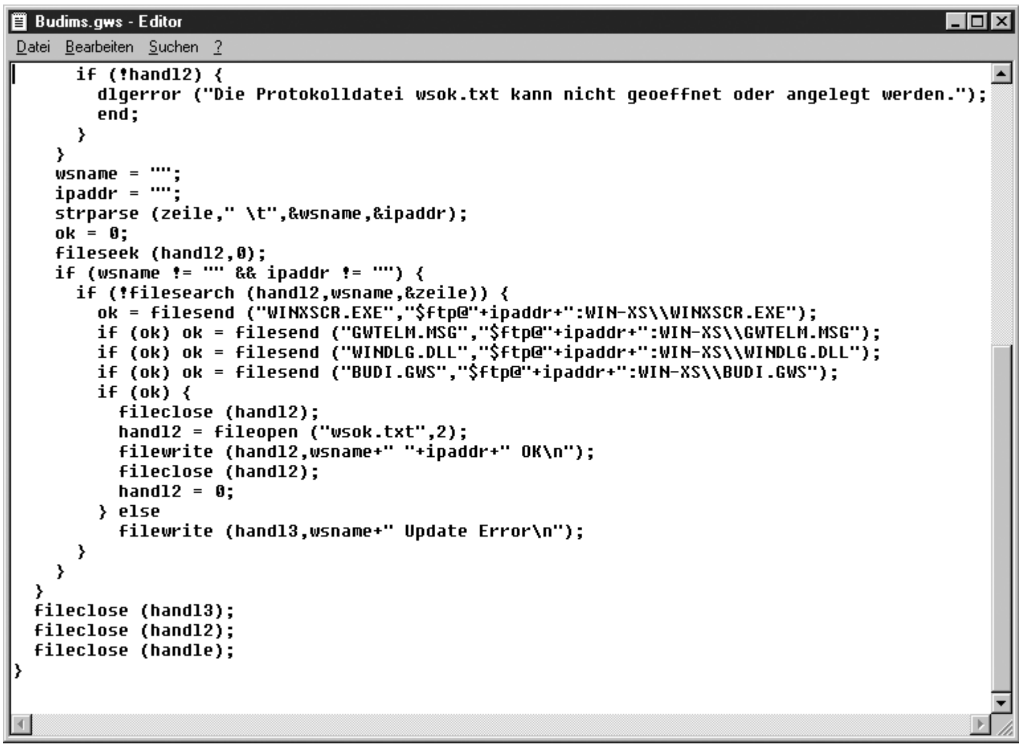


Abbildung 3.11: Infra-XS Konfiguration

Trotzdem fällt positiv auf, dass sämtliche Einstellungen zentral von einer Stelle aus erledigt werden können. Wie vorher schon erwähnt, werden Messungen von den *XS-Agenten* unternommen und zum Server geschickt. Dabei wird zum Teil auf der Eingabeoberfläche einer Applikation, wie auch auf der Netzwerkbasis gemessen. Win-XS (Teilkomponente der *XS-Agenten*) misst dabei auf der Anwendungsoberfläche, indem eine Applikation automatisiert nach einem aufgenommen Script bedient und mittels entsprechender Messpunkte in Antwortzeiten übersetzt wird. Diese Markierungen befinden sich innerhalb des Skriptes und kennzeichnen relevante Transaktionsbereiche. Die einzelnen Markierungen können vereinfacht als START und STOP Signale verstanden werden, die über Antwortzeitverhalten der jeweiligen Applikation Auskunft gibt. D.h. ein Programm wird nach einem vordefinierten Ablaufplan benutzt, wobei ein Script jeweils den nächsten Schritt angibt. Zwischen diesen Schritten können Markierungen gesetzt werden, die bei der Abarbeitung eine Zeiterfassung auslösen. Die Zeit die eine Applikation braucht um von einer Markierung zur nächsten zu gelangen kann als Antwortzeitverhalten ausgelegt werden. Das Script hat dabei das Aussehen eines C ähnlichen Sourcecodes, wie auf Abb. 3.12 zu sehen ist.


Budims.gws - Editor
Datei Bearbeiten Suchen ?

```
      if (!handl2) {
        dlgerror ("Die Protokolldatei wsok.txt kann nicht geoeffnet oder angelegt werden.");
        end;
      }
    }
    wsname = "";
    ipaddr = "";
    strparse (zeile," \t",&wsname,&ipaddr);
    ok = 0;
    fileseek (handl2,0);
    if (wsname != "" && ipaddr != "") {
      if (!filesearch (handl2,wsname,&zeile)) {
        ok = filesend ("WINXSCR.EXE","$ftp@"+ipaddr+":WIN-XS\\WINXSCR.EXE");
        if (ok) ok = filesend ("GWTELM.MSG","$ftp@"+ipaddr+":WIN-XS\\GWTELM.MSG");
        if (ok) ok = filesend ("WINDLG.DLL","$ftp@"+ipaddr+":WIN-XS\\WINDLG.DLL");
        if (ok) ok = filesend ("BUDI.GWS","$ftp@"+ipaddr+":WIN-XS\\BUDI.GWS");
        if (ok) {
          fileclose (handl2);
          handl2 = fileopen ("wsok.txt",2);
          filewrite (handl2,wsname+" "+ipaddr+" OK\n");
          fileclose (handl2);
          handl2 = 0;
        } else
          filewrite (handl3,wsname+" Update Error\n");
      }
    }
  }
  fileclose (handl3);
  fileclose (handl2);
  fileclose (handle);
}
```

Abbildung 3.12: Beispielscript aus [Gey]

Die Syntax der Sprache entspricht einer eigenen Makroprogrammiersprache der Firma G&W, daher auch GW-Tel Makroprogrammiersprache genannt. *Frame-XS* die zweite Komponente der *XS-Agenten*, ist wie der Name schon sagt, ein auf der Netzwerkschicht operierendes Messprogramm. Innerhalb des von Win-XS vorgegebenen Zeitfensters, analysiert Frame-XS ein- und ausgehende Pakete auf der TCP/IP Schicht und kombiniert diese Ergebnisse mit den Zeiten von Win-XS. Dabei werden relevante Daten nach Verbindungsart mit Quell- und Zieladresse, wie auch verbindungsspezifische Ereignisse, die in Korrelation zur überwachenden Anwendung stehen, wie z.B. Netzlaufzeit eines Paketes, erfasst. Die Kombination von Frame-XS und Win-XS erlaubt anschließend die Aufsplittung der Gesamtantwortzeit einer End-to-End Kommunikationsbeziehung zwischen Endgerät und Host- oder Serversystem, in PC-, Netz- und Hostanteil. Abb. 3.13 veranschaulicht diese Aufsplittung anhand einer detaillierten Beschreibung eines Browserverhaltens bei einem Webseitenaufruf. Der Browser sucht nach der Eingabe der URL, zuerst innerhalb seines Caches nach dieser Seite und versucht erst dann eine direkte Verbindung herzustellen. Dies hat mit der eigentlichen Transaktion, nämlich dem Download der Seite , nichts zu tun. Daher verwendet *INFRA-XS* eine Kombination, um zum einen die Transaktion in seiner Gesamtheit, wie auch in seinen Einzelbereichen zu messen. Die Einzelbereiche sind in der Abb. 3.13 durch die verschiedenen Balken markiert. Zeit zwischen zwei kurzen Balken gehört dabei zum Netzanteil der Transaktion, wobei die Zeit zwischen einem längeren und einem kürzeren Balken, zum PC- oder Hostanteil gerechnet wird.

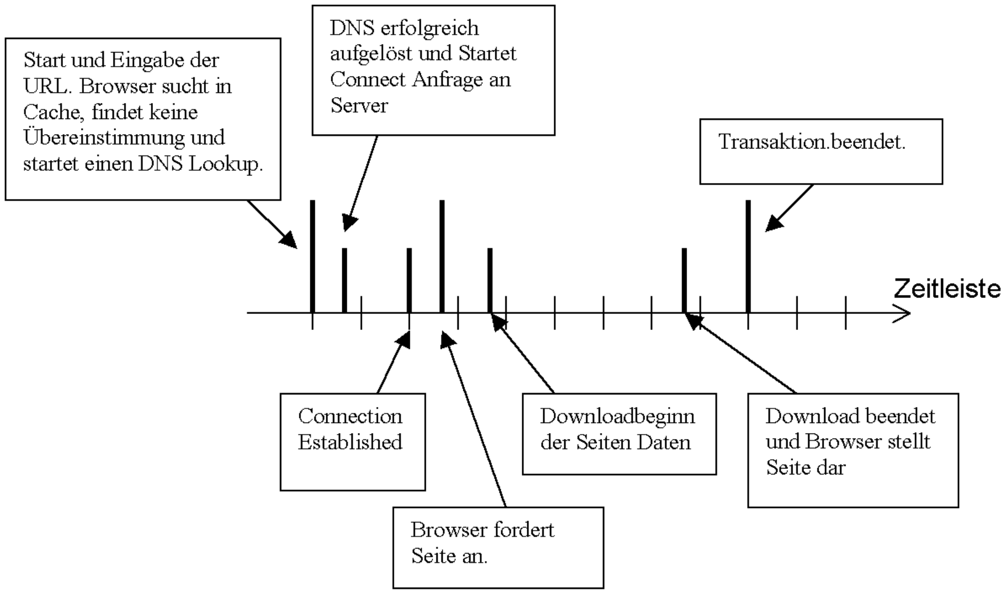


Abbildung 3.13: Aufsplittung in PC- Netz- und Hostanteil

Nun noch zu einem kleinen Reportbeispiel. Abb. 3.14 zeigt einen webbasierten Statusreport einer Mailüberwachung. Reports dieser Art können mit Hilfe der *XS-Clients* und dem darin enthaltenen XS-AuswertungsDesigner erzeugt werden. Auf dieser Webseite sieht man eine Anzahl von E-Mail Server die von einigen Agenten überwacht werden. Für jeden Server werden die geschickten und empfangenen E-Mails und ihre Ausfallzeiten, sowie die Transferdauer aufgelistet.

3.3.5 Kosten

Die Kosten für *Infra-XS* bewegen sich in folgenden Größenordnungen:

- 50 Agenten 250.000 DM
- 200 Agenten 600.000 DM

3.3.6 Einsatz

Infra-XS wird bereits erfolgreich bei folgenden Firmen eingesetzt:

- Commerzbank, wobei hier hauptsächlich das Online Banking überwacht wird
- RV
- BASF

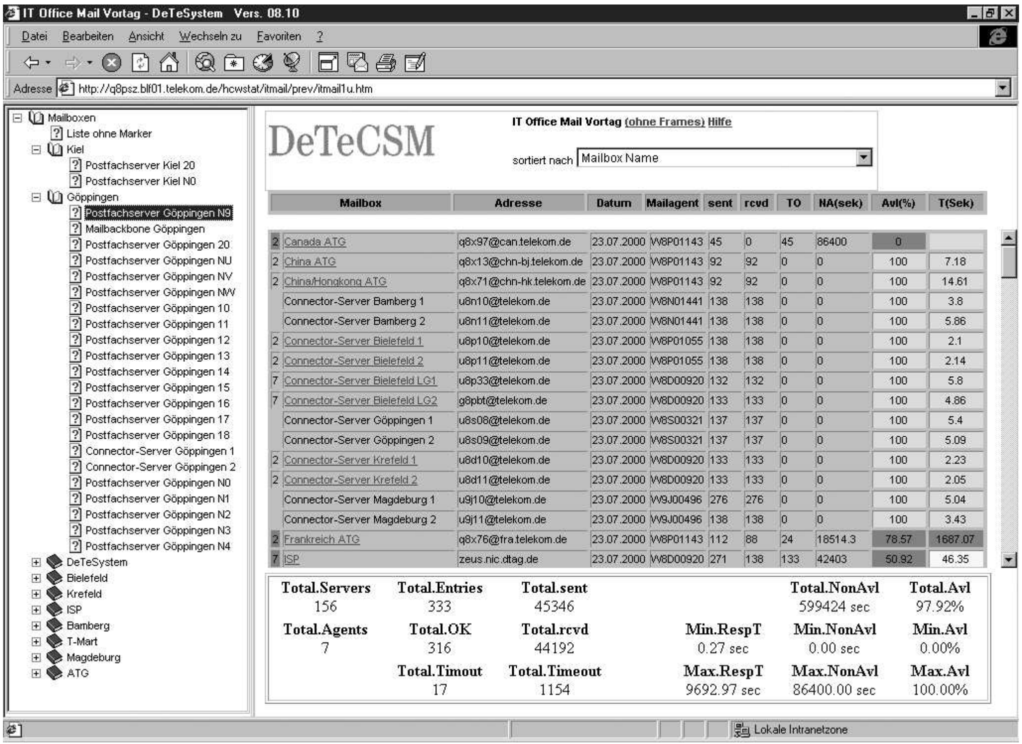


Abbildung 3.14: Beispielreport einer Mailüberwachung von Infra-XS

- T-Mobil
- Telekom, DeTeCSM überwacht damit z.B. den Mail-Backbone-Bereich verschiedenster Kunden

3.4 ARM 3.0 der ARM Working Group

ARM steht als Abkürzung für *"Application Response Measurement"* und ist ein aufkommender Standard zur Messung der Performance und Verfügbarkeit von Geschäftstransaktionen und deren Komponenten. Was als ARM API im Juni 1996 veröffentlicht, begann als separates und unabhängiges Projekt bei Tivoli und HP. 1997 erschien bereits der Nachfolger ARM 2.0 und war nun ein Gemeinschaftsprojekt vieler Unternehmen , die sich zur ARM Working Group zusammenschlossen. ARM wurde schließlich im Juli 1998 als Standard von der *Open Group*, als Teil der *IT Dial Tone*TM Initiative, zugelassen

3.4.1 Funktionsweise

Die ARM API ist eine einfache API mit deren Hilfe Applikationen, wesentliche Informationen über Transaktionen, an einen Agenten übermitteln können. Die Applikation muss lediglich die ARM API vor dem Start und nach Beendigung einer Transaktion (oder Subtransaktion) aufrufen. Unterstützt wird die API durch einen Agenten der die Zeit zwischen den Start und Stop Aufrufen berechnet und sie für eine Management Station verfügbar macht. Die ARM API beinhaltet dabei nur wenige Aufrufe:

ARM API Funktionen in Java Syntax

ArmTranDefinition	Repräsentiert die Definition eines Transaktionstyps. Erzeugt und initialisiert ein ArmTranDefinition Objekt.
ArmTranDefinition.setApplName()	Registriert die statischen Attribute einer Transaktion, wie z.B. dessen Namen
ArmTranDefinition.setTranName()	z.B. Den Transaktionsnamen
ArmTranDefinition.setTranId()	z.B. Die TransaktionsID. (Liste nicht vollständig)
ArmTransaction.start()	Zeigt an, dass der Start einer Instanz einer Transaktion begonnen hat
ArmTransaction.update()	Ein Heartbeat der anzeigt, dass eine Transaktionsinstanz noch immer ausgeführt wird. Angabe ist optional.
ArmTransaction.Stop()	Zeigt an, dass eine Transaktionsinstanz beendet ist
ArmTransaction.completeTransaction()	Kann statt Start-Stop Aufrufe benutzt werden um Antwortzeiten an ARM Agent zu übermitteln. Messung der Antwortzeiten muss die Applikation selbst übernehmen.

Abb. 3.15 zeigt, wie Business Applikationen von der ARM API überwacht werden. Um dabei das gesamte End-to-End Bild einer Transaktion zusammenzubauen, werden neben den Start und Stop Befehlen sogenannte Korrelatoren (ein ca. 30 Byte großes Datenfeld) mitgeschickt. Eine Clientanwendung fordert einen Korrelator an, der dann sowohl an den ARM Agenten, als auch an den Applikationsserver übertragen wird. Die Applikation des Servers übernimmt diesen Korrelator und reicht diesen mitsamt seinen ARM Start und Stop Aufrufen an den ARM Agenten weiter, der diese Informationen wiederum an eine Managerstation weiterleitet. Dort kann dann ein Gesamtbild der Transaktionszeit gebildet werden, egal wie viele Zwischenstationen die Transaktion durchlaufen hat.

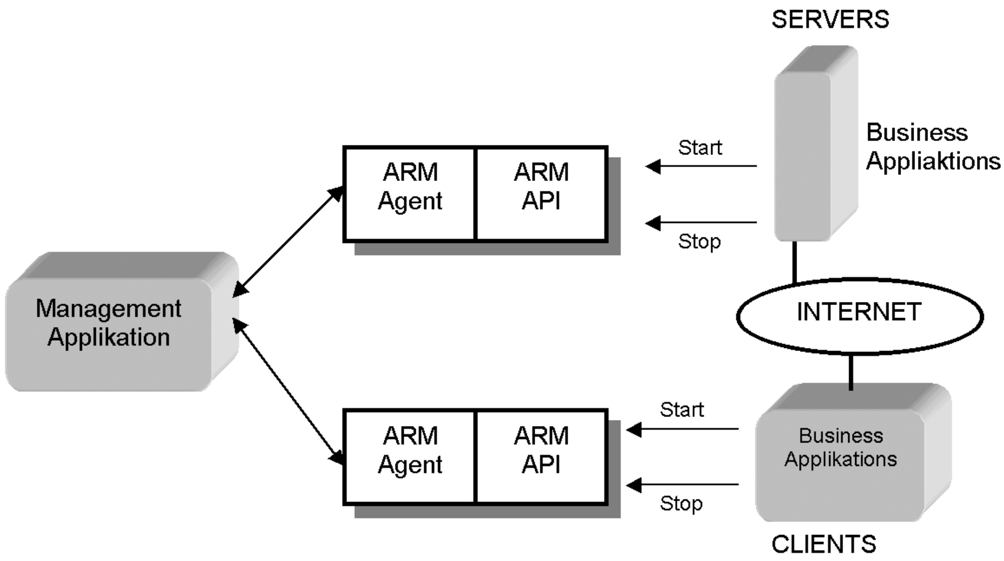


Abbildung 3.15: ARM API Überblick aus [Johb]

Besonderheit in der ARM Version 3.0 ist die neue JAVA Funktionalität. ARM 3.0 ist die erste Version mit Unterstützung der *Native Java Bindings*. Zusätzlich enthält ARM 3.0 einige neue Aufrufe, wie z.B. `arm_complete_transaction()`. Damit ist es möglich einen asynchronen ARM Aufrufe pro Transaktionsinstanz zu erzeugen. Normalerweise müssen Start und Stop Aufrufe innerhalb zu messender Transaktionsbereiche erfolgen. Können diese aufgrund programmiertechnischer Feinheiten jedoch nicht erlaubt werden, kann es zu unkorrekten Messungen kommen. Der `arm_complete_transaction()` Aufruf, wird nur ein einziges mal an einer beliebeigen Stelle innerhalb der Transaktion aufgerufen und man benötigt so nicht mehr einzelne Start und Stop Aufrufe der API. Die Applikation selbst übernimmt die Messung und reicht sie mittels der `arm_complete_transaction()` an den Agenten weiter. Kritische Programmstrukturen können so ungestört von ARM Aufrufen bleiben.

3.4.2 Beipiel

Folgendes Beispiel in Java Syntax soll den Aufbau und Funktionsweise einer ARM Implementierung etwas näher erklären.

```
// UUID definieren
byte idDiskBackup[] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};

// Transaktions ID kreieren
ArmTranId tidDiskBackup = new ArmTranId(idDiskBackup);

// Transaktions Definition kreieren
ArmTranDefinition tdDiskBackup = new ArmTranDefinition();
tdDiskBackup.setApplName("Disk Backup Software");
tdDiskBackup.setTranName ("Backup Disk");
tdDiskBackup.setTranId(tidDiskBackup);
tdDiskBackup.register();
```

```
// Transaktion kreieren
ArmTransaction tDiskBackup = new ArmTransaction(tdDiskBackup);

// Transaktion ausführen
tDiskBackup.start();            //Starten des Backups
try
{
  for (int i=0; i<diskCount; i++) // i Disk's sichern
  {
     // Backup disk[i];
     tDiskBackup.update();      // weitere Disk gesichert
  }
  tDiskBackup.stop(ArmTransaction.ARM_GOOD);  //alle Disks erfolgreich gesichert
}
catch (Exeption e);
{
   // Ausnahmebehandlung abfangen
   tDiskBackup.stop(ArmTransaction.ARM_FAILED); // Transaktion nicht erfolgreich
}
```

Der erste Schritt in der Implementierung von ARM Befehlen sieht die eindeutige Identifizierung der Transaktion vor. Empfohlen wird eine UUID (Universal unique ID). Unter Microsoft Windows auch als GUID (Global Unique ID) bekannt. Mit dieser UUID (16-Byte langen Wert) kann die Transaktion weltweit eindeutig identifiziert werden. Die Funktion `ArmTranId()` mit der UUID (hier: `idDiskBackup`) erzeugt die Transaktions ID `tidDiskBackup`.

Zusätzlich können noch weitere Informationen an die Transaktions ID gebunden werden. Mit Hilfe der `ArmTranDefinition()` können neben Applikationsnamen und Transaktionsnamen auch noch weitere Metriken der ID zugeordnet werden um z.B. File Größe oder Usernamen in Auswertungen mit einfließen zu lassen. Unser Beispiel beschränkt sich hier jedoch lediglich auf Applikatinsname (`tdDiskBackup.setApplName (''Disk Backup Software'');`) und Transaktionsname (`tdDiskBackup.setTranName (''Backup Disk'');`). Ist die Transaktion definiert kann ein Transaktionsobjekt generiert werden. Hier "`ArmTransaction tDiskBackup = new ArmTransaction(tdDiskBackup);`".

Wird nun die eigentliche Transaktion gestartet muss ein Startsignal (`tDiskBackup.start();`)an den ARM Agenten geschickt werden um eine Messung einzuleiten. Gleichzeitig werden die zusätzlichen Informationen wie Applikationsname und Transaktionsname an den Agenten übermittelt. Läuft eine Transaktion über einen längeren Zeitraum, wie hier ein Disk Backup, kann mit Hilfe der Update Funktion (`tDiskBackup.update();`) gekennzeichnet werden, dass diese noch immer läuft und noch nicht abgestürzt ist. Ist die Transaktion schließlich beendet, wird ein Stop Signal an den ARM Agenten geschickt (`tDiskBackup.stop(ArmTransaction.ARM_GOOD);`).

Der Agent berechnet nun die Zeit zwischen *ARM_Start* und *ARM_Stop* und erhält so das Antwortzeitverhalten der Applikation. Weiter Funktionalität, wie Darstellung, Speicherung, Korrelation übernimmt dabei der ARM Agent.

Neuerung in der ARM 3.0 Version stellt die `completeTransaction()` Funktion dar. Wie oben schon erwähnt wurde damit die Möglichkeit geschaffen kritische Programmstrukturen unangetastet zu lassen und den Agentenaufruf beliebig zu positionieren. Auch hierzu ein kleines Beispiel.

```
// Transaktion
```

```
try
{
  startTime = System.currentTimeMillis();
  for (int i=0; i<diskCount; i++) // i Disks sichern
  {
    // Backup Disk i;
  }
  stopTime = System.currentTimeMillis();
  respTime = stopTime - startTime;
  // respTime wird in Nanosekunden angegeben, also mit 10**6 malnehmen
  tDiskBackup.completeTransaction(ArmTransaction.ARM_GOOD,respTime*1000000
                                  , stopTime);
}
catch (Exeption e)
{
  stopTime = System.currentTimeMillis();
  tDiskBackup.completeTransaction(ArmTransaction.ARM_FAILED, 0, stopTime);
}
```

Wie in diesem Beispiel zu sehen, fehlen die Start und Stop Aufrufe wie im letzten Beispiel gänzlich. Trotzdem wird auch hier eine Zeitmessung durchgeführt. Diesmal übernimmt jedoch die Applikation selber die Zeitmessung, indem vor dem eigentlichen Beginn der Transaktion, dem `Disk Backup`, ein Zeitstempel abgespeichert wird. Diese Zeit wird in der Variablen `startTime` zwischengespeichert und nach Beendigung der Transaktion von der `stopTime` abgezogen. Die resultierende Zeit `respTime` spiegelt die Dauer der Transaktion wieder und wird durch die neue ARM Funktion `.completeTransaction()` an den ARM Agenten übergeben. Somit ist man in der Lage auch Applikationen mit kritischen Programmstrukturen, in denen z.B. aus programmiertechnischen Gesichtspunkten keine API Aufruf erlaubt sind, mittels ARM exakt zu überwachen.

3.4.3 Zusammenfassung

ARM erscheint auf den ersten Blick als die perfekte Lösung um Antwortzeiten auf Applikationsebene zu ermitteln. Folgende Punkte sind dabei besonders zu erwähnen.

Alleine durch den direkten Ansatzpunkt der Messungen (im Sourcecode verankerter ARM Code) erhalten die Ergebnisse eine erhöhte Aussagekraft gegenüber anderen Produkten. Dies wird deutlich wenn man bemerkt das diese hauptsächlich durch sekundäre Merkmale, wie Oberflächenevents oder Netzverkehr, das Antwortzeitverhalten der Anwendung bestimmen.

Daneben kann man durch intelligente Implementierung auch Aufschluss über die Dauer von logisch getrennten Untertransaktionen einer Transaktion erhalten. Ein Beispiel dazu. Angenommen eine Transaktion setzt sich aus 3 logisch getrennten Einzeltransaktionen zusammen. DNS Auflösung, ServerConnect und Download. Bei expliziter Implementierung und Beachtung der einzelnen Teilbereiche der Transaktion, können diese getrennt erfasst und gemessen werden. Diese erhöhte Analysegrundlage kann eine wichtige Entscheidungshilfe für Lastbalancierung oder Verfügbarkeitsanforderungen darstellen.

Weiter lassen sich durch Korrelatoren Transaktionen die die Ursache einer vorhergehenden Transaktion sind, als zusammengehörig identifizieren. Diese Informationen können z.B. benutzt werden um Routen zu optimieren oder um Fehlerquellen aufzuspüren.

Trotz der überwältigenden Vorteile von ARM ist ein Einsatz schwierig zu realisieren. Wie bereits unter den Konzepten im Punkt Implementierung zu ersehen, stehen dem praktischen Einsatz nicht zu unterschätzende wirtschaftliche Hürden im Weg.

Diese sind hoher finanzieller und zeitlicher Aufwand im Rahmen einer nachträglichen Implementierung. Dieser Aufwand wird besonders ersichtlich, werden Größe, Vernetzung und Spezialisierung von Firmensoftware in Betracht gezogen. Darüber hinaus kann sich ein erhöhtes Risiko an Ausfällen aufgrund fehlerhafter Implementierung ergeben.

Obwohl ARM viele hervorragende Eigenschaften besitzt, schrecken die hohen Anforderungen vor einem Einsatz ab. Würde Software bereits in der Entwicklung mit ARM Komponenten ausgestattet, ergäbe sich sehr wahrscheinlich ein weit besseres Bild.

Kapitel 4

Resultate

Wie man gesehen hat, gibt es die unterschiedlichsten Ansätze und Produkte um eine Antwortzeitüberwachung zu realisieren. Im folgenden werden die Konzepte und ihre Produkte anhand des Bewertungskataloges zusammengefasst.

4.1 Etewatch

Etewatch, als Vertreter der Eingabeoberflächenbasierten Auswertung, bewies durch die leichte und unkomplizierte Installation, wie einfach es sein kann eine Antwortzeitüberwachung auf Applikationsebene durchzuführen. Ein wesentliches trug auch die ausführliche und verständliche online Dokumentation dazu bei. **Installationsaufwand und Systemanforderungen** sind zwar positiv zu bewerten, jedoch muss darauf hingewiesen werden, daß eine Installation auf jedem Clientsystem durchgeführt werden muss, was einen erheblichen Zeitaufwand bedeutet. Einen weiteren Pluspunkt im Bereich **Aussagekraft** erhält das Produkt aufgrund der Tatsache, dass Messungen direkt an der relevanten Anwendung unter der Benutzung eines realen Anwenders passieren. Die so gewonnenen Ergebnisse spiegeln nämlich die realen *"Wartezeit"* Erfahrungen eines Anwenders wieder und gelten gerade deswegen als besonders aussagekräftig. Ein Manko des ganzen liegt jedoch am Konzept selbst. Durch die Analyse von Oberflächenevents erhält man keine tieferen Einblicke in das Transaktionsverhaltens an sich um z.B. Diagnosen bei Problemfällen stellen zu können. Der mäßige **Detaillierungsgrad der Messdaten** ist daher negativ zu bewerten. Der Kostenfaktor des Produkts ist eher positiv zu werten, da er im Gegensatz zu den anderen Produkten weitaus geringer ist. Geht man von der Überwachung von ca. 200 Computern aus, ergibt sich bei Etewatch ein Preis von ca. 82.000 DM. Der allgemeine Eindruck wurde leider durch Fehlmessungen getrübt, die durch ungenaue Eventüberwachung der Applikation auftraten. Trotzdem stellt sich *Etewatch* als innovativer und erfolgversprechender Ansatz der Applikationsorietierten Antwortzeitüberwachung vor.

4.2 Firehunter

Das Produkt *Firehunter* von Agilent arbeitet nach dem Verfahren der Simulierten Anfragen indem proprietäre Software vordefinierte Serviceanfragen von wenigen festen Standorten aus durchführt und diese im einzelnen misst. Damit wird die Forderung nach **Aussagekräftigen Messungen**, d.h. eine Messung beim Benutzer selbst, nicht erfüllt und damit negativ bewertet. Trotzdem hat diese Art der Antwortzeitüberwachung auch positive Eigenschaften. Besonderer Vorteil dieser Methode ist, daß Problemfälle frühzeitig von einem dedizierten Mess-PC erkannt werden und nicht erst von einem menschlichen Benutzer.

Installationsaufwand und Systemanforderungen müssen etwas differenzierter betrachtet werden. Während der Installationsaufwand sehr gering ist, d.h. es müssen nur einige wenige Rechner installiert werden, sind die Systemanforderungen an diese Rechner entsprechend hoch. Neben dem geringen Installationsaufwand bietet *Firehunter* durch die große Anzahl von vorgefertigten Templates für Überwachungsparameter, eine äußert leichte Bedienung und Konfiguration. *Firehunter* bedient sich wie *Etewatch* einer komfortablen online Hilfe, um dem Benutzer das Zurechtfinden so leicht wie möglich zu machen. Störend wirken sich nur die sehr hohen Systemvorraussetzungen aus (empfohlen wird 256MB RAM, PII 500MHz für jede Überwachungskomponente). Diese bedeuten hohe Kosten bei großflächigem Einsatz. **Installationsaufwand und Systemanforderungen** lassen somit zusammengenommen weder eine deutlich positive noch negative Wertung zu.

Die Einsatzkosten für einen mittleren Betrieb mit ca. 200 Rechnern kann überschlagsmäßig mit 120.000 DM angesetzt werden. Davon werden 80.000 DM für das Firehunter Grundpaket verbucht und 40.000 DM für Hardware. Damit liegen die Kosten für den Einsatz leicht über dem von Etewatch und lassen so die Bewertung der **Einsatzkosten** etwas schlechter ausfallen.

Im Punkt **Detaillierungsgrad der Messdaten** erhält Firehunter eine wesentlich positivere Bewertung als sein Vorgänger. Durch den Einsatz von proprietärer Software ist *Firehunter* in der Lage neben kompletten Transaktionen auch Teiltransaktionen zu messen(z.B. FTP Transaktion: FTP-Connect und FTP-Download).

Insgesamt kann man sagen, dass es sich bei dem Produkt *Firehunter* um eine gelungene Realisierung des Konzepts über simulierte Anfragen handelt, mit etwas zu hohen Anforderungen an die Hardware.

4.3 Infra-XS

Infra-XS benutzt wie Firehunter ein System von automatisierten einzelnen Rechnern, die Anfragen an Serversysteme durchführen. Ähnlich wie bei Firehunter erfüllt auch Infra-XS die Forderung nach Nähe und**Aussagekraft** nicht und wird dementsprechend negativ bewertet. Im Bereich **Installationsaufwand und Systemanforderungen** ergeben sich weitere Minuspunkte. Obwohl Assistenten bei der Installation helfen, wird man anschließend doch von den komplexen und umfangreichen Konfigurationsparametern erschlagen. Dazu kommt, daß für die Überwachung eines Dienstes, zuerst ein Script in einer eigenen Programmiersprache entworfen werden muss, daß dann eine benutzerähnliche Bedienung einer Applikation durchführt. Eine sehr komplizierte Vorgehensweise, die meistens die fachkundige Hilfe des Herstellers benötigt. Damit verbunden sind natürlich zusätzliche Kosten bei Installation und Wartung. Dazu kommen noch Kosten für die Software selbst, die für ein mittelständisches Unternehmen bei ca. 200.000 DM liegen können. Zusätzlich müssen laufende Kosten für Wartung und Anpassung der Agenten durch den Hersteller mit einkalkuliert werden. In Punkto **Einsatzkosten** schneidet Infra-XS daher noch schlechter ab als Firehunter. Im Gegensatz dazu liegt der **Detaillierungsgrad der Messdaten** jedoch um einiges höher. Durch den kombinierten Einsatz von allgemeiner Transaktionsmessung und Netzorientierter Verbindungs-und Paketanalyse lässt *Infra-XS* einen tieferen Blick in die überwachte Transaktion und ihrer Subtransaktionen zu als die bisher genannten Konkurrenzprodukte.

Allgemein ist *Infra-XS* ein schwieriges und komplexes Monitoring Tool, an dessen Handhabung sich nur geschultes Fachpersonal wagen sollte.

4.4 ARM 3.0

ARM 3.0 übertrifft in manchen Aspekten die anderen Produkte, hat dafür aber auch einige gravierende Nachteile. Positiv zu werten sind folgende. Der **Detaillierungsgrad**, sowie die **Aussagekraft** der Messungen, sind dabei besonders hervorzuheben. Grund dafür ist der ARM setzt bereits im Sourcecode der Applikation an und erlaubt somit absolut konkrete und **direkte** und damit Messungen. Des weiteren lässt sich mit Hilfe von ARM eine ausgesprochen genaue und tiefe Einsicht in Transaktionen vornehmen. Bei entsprechender Implementierung lassen sich Zeiten jeder Subtransaktion messen und auswerten. Damit erhält man gleichzeitig ein Analysewerkzeug, mit dem Schwachstellen im Netz identifiziert werden können. Im Gegensatz zu den anderen Produkten lassen sich mit ARM verteilte Transaktionen mittels Korrelatoren zusammenfassen und in Abhängigkeit bringen. Auch dies kann als eine wichtige Analysefunktion angesehen werden, die den anderen weitgehend fehlt.

Dem gegenüber stehen Zeit und Kostenaufwand für Implementierung. Obwohl ARM nur wenig API Aufrufe verwendet, müssen diese an relevanten Stellen im Sourcecode der Anwendung platziert werden. Fehlt detailliertes Wissen über den Sourcecode oder ist dieser zu komplex, ist eine Implementierung schwierig oder sogar unmöglich. ARM erhält daher eine eher negative Wertung im Bereich **Installationsaufwand**. Über **Einsatzkosten** können leider keine genauen Aussagen gemacht werden, da es sich bei ARM nicht um ein Werkzeug handelt, sondern vielmehr um einen API-Standard für Antwortzeitmessungen durch Anwendungsinstrumentierung. Während der API-Standard im Internet von der *ARM Working Group* als SDK kostenlos zur Verfügung gestellt wird, ist im Gegensatz dazu, der Erwerb der Agenten kostenpflichtig. Diese Kosten sind jedoch von Hersteller zu Hersteller unterschiedlich und machen damit eine genau Aussage über **Einsatzkosten** unmöglich. Diese wirtschaftlichen Aspekte machen den Einsatz einer Überwachung mittels ARM entsprechend schwer und teuer. Trotzdem ist, vom theoretischen Standpunkt aus, dieser Ansatz am meisten erfolgversprechend für effektive und professionelle Antwortzeitüberwachung in der Zukunft.

Folgende Tabelle soll die Ergebnisse zusätzlich grafisch veranschaulichen.

	Infra-XS	Firehunter	Etewatch	ARM 3.0
Aussagekraft	-	-	+	++
Installationsaufwand	- -	+	+	- -
Einsatzkosten	0	+	+	0
Detaillierungsgrad	+	+	- -	++

Literaturverzeichnis

[Agi] Agilent Technologies, http://firehunter.comms.agilent.com/library/manage.htm. *Measurement and Management of Internet Services Using Agilent Firehunter.*

[Cana] Candle Corporation. *ETEWatch.*

[Canb] Candle Corporation, http://www.candle.com/publications/product_documentation/index.html. *ETEWatch FactSheet.*

[Doc] Online Document. *INFRA-XS Online Documentation.* Geyer & Weinig, http://www.gwtel.de/html-de/home/index_ms.html.

[Gey] Geyer & Weinig. *INFRA-XS Benutzerhandbuch.*

[HAN99] H.-G. Hegering, S. Abeck, and B. Neumair. *Integrated Management of Networked Systems – Concepts, Architectures and their Operational Application.* Morgan Kaufmann Publishers, ISBN 1-55860-571-1, 1999. 651 p.

[HR00] R. Hauck and I. Radisic. Monitoring Application Service Performance – Classification and Analysis of Existing Approaches. In *Workshop of the OpenView University Association (OVUA 2000)*, June 2000.

[Joha] Mark W. Johnson. *The Application Response Measurement (ARM) API, Version 2.* Tivoli Systems, http://www.cmg.org/regions/cmgarmw/marcarm.pdf.

[Johb] Mark W. Johnson. *ARM 3.0 - Enabling Wider Use of ARM in the Enterprise.* Tivoli Systems, http://www.cmg.org/regions/cmgarmw/arm30enhancements.pdf.

[Tiv] Tivoli Systems, http://www.cmg.org/regions/cmgarmw/JavaARM 00.55.zip. *Java Bindings for ARM.*