



Forschungszentrum Karlsruhe
in der Helmholtz-Gemeinschaft



Entwicklung eines Nagios-Plugins zur Überwachung und Auswertung von Funktionen und Fehlern in Content-Management-Systemen

BACHELORARBEIT

für die Prüfung zum
Bachelor of Engineering

des Studienganges

Informationstechnik

an der Dualen Hochschule Karlsruhe

von

Andreas Paul

Bearbeitungszeitraum: 25.05.2009 – 23.08.2009

Matrikelnummer: 108467

Kurs: TIT06GR

Praxissemester: 6

Ausbildungsfirma:
Forschungszentrum Karlsruhe GmbH (FZK)
Steinbuch Center for Computing
Hermann-von-Helmholtz-Platz 1
76344 Eggenstein-Leopoldshafen

Betrieblicher Betreuer: Dr. Doris Wochele

Prüfer der DH Karlsruhe: Herr Holger Raff

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbst angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher keiner Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ich versichere hiermit wahrheitsgemäß, die Arbeit bis auf die dem Aufgabensteller bereits bekannte Hilfe selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderung entnommen wurde.

Karlsruhe, den 21. August 2009

.....
Ort, Datum

(Andreas Paul)

Abstract

Dokumenten-Management-Systeme (**DMS**) bilden eine zentrale Dienstleistung im Karlsruhe Institute of Technology (KIT). Diese Systeme sind komplex aufgebaut und müssen, um einen stabilen Betrieb zu ermöglichen, mit ausgereiften Überwachungsroutinen getestet werden. Zum gegenwärtigen Zeitpunkt gibt es keine Überwachungslösung im KIT, die explizit diese Aufgabe zufriedenstellend erfüllt.

Diese Bachelorarbeit beschreibt die Entwicklung von neuen Werkzeugen für die Open Source-Überwachungssoftware Nagios um das in KIT verwendete Dokumenten-Management-System **Oracle UCM**¹ auf Fehlverhalten hin zu kontrollieren. Diese sogenannten Plugins lassen sich in bestehende Nagios-basierende Systeme einbinden und erweitern deren Bandbreite an zu überwachenden Elementen. Das Hauptaugenmerk lag, auf der Simulation von Benutzerverhalten und der Erkennung der dabei auftretenden Fehler. Die richtige Interpretation der Fehler liefert den verantwortlichen Personen wichtige Hinweise auf die tatsächliche Ursache und erleichtert die Lösungsfindung.

Document-management-systems present a vital service in the Karlsruhe Institute of Technology. The complexity of these systems demands a sophisticated monitoring solution to provide a reliable operation of this service. At present in time there are no satisfying solutions available for this task.

This thesis describes the development of a tool to monitor the document-management-system **Oracle UCM**, which is used in KIT, in combination with the open source software Nagios. These so-called plugins can be integrated in existing Nagios-based systems and can thus expand the range of controlled objects. The main focus was on the simulation of typical user-actions and the detection of thereby resulting errors. The correct interpretation of these

¹Oracle Universal-Content-Management



ABSTRACT

2

errors will indicate the reason of the problem and thus help the responsible persons finding a possible solution.



Inhalt

1 Einleitung	5
2 Aufgabenstellung	7
3 Grundlagen	9
3.1 Überwachungssysteme	9
3.1.1 Ressourcenbelastung	10
3.1.2 Netzwerkstruktur und Abhängigkeiten	11
3.1.3 Sicherheitsaspekte	13
3.2 Dokumenten-Management-Systeme	14
3.2.1 Eingabe	18
3.2.2 Verwaltung und Archivierung	20
3.2.3 Ausgabe	20
3.3 Content-Management-Systeme	22
3.4 Enterprise-Content-Management-Systeme	24
3.5 Serviceorientierte Architektur	24
3.6 Web Services-Architektur	26
4 Nagios	30
4.1 Allgemein	30
4.2 Aufbau	31
4.3 Überprüfungsmethoden	37
4.3.1 Aktive Checks	37
4.3.2 Passive Checks	37
5 Oracle UCM	43
5.1 Allgemein	43
5.2 Aufbau	43
5.3 Konkrete Verwendung	46
6 Überwachungselemente	47
6.1 Statusabfragen	47



6.2	Überwachung der Funktionalität	48
6.3	Auswerten von Logdateien	49
6.4	Benutzersimulation	50
6.5	Zusammenfassung	51
7	Umsetzung	53
7.1	Aufbau der Testumgebung	53
7.2	Auswahl der geeigneten Überwachungsmethode	53
7.3	Übersicht Nagios-Agenten	54
7.3.1	Unix-Agenten	54
7.3.2	Windows-Agenten	57
7.3.3	Auswahl und Konfiguration des Nagios-Agenten	59
7.4	Umsetzung der Systemüberwachung	63
7.5	Umsetzung der Funktionalitätstest	64
7.6	Auswertung der Logdateien	66
7.7	Benutzersimulation	67
8	Ergebnis	75
9	Zusammenfassung	78
10	Ausblick	81

1 Einleitung

Mit dem Zusammenschluss des Forschungszentrum Karlsruhe und der Universität Karlsruhe (TH) zum Karlsruhe Institute of Technology ist eine Einrichtung mit 8000 Wissenschaftlern und Mitarbeitern, 18000 Studierenden und circa 300 externen Mitarbeitern und Gästen entstanden.

Die IT-Infrastruktur für den organisatorischen und wissenschaftlichen Betrieb liegt in der Verantwortung des Steinbuch Center für Computing (SCC), das aus der Verschmelzung des Rechenzentrums der Universität und dem Institut für Wissenschaftliches Rechnen (IWR) hervorgegangen ist.

Für alle Schichten der IT-Infrastruktur und alle angebotenen Dienstleistungen muss der Betrieb durch das Rechenzentrum überwacht werden. Die Überwachung des Dokumenten-Management-System, einer zentralen Dienstleistung, ist Ziel dieser Arbeit.

Die Hauptaufgaben eines Dokumenten-Management-Systems sind die zentrale Speicherung, Bearbeitung und Verwaltung von Dokumenten. Dabei können diese Dokumente Dateien in unterschiedlicher Form sein wie Microsoft Word Dateien, Excel Tabellen, Dateien im Portable Document Format ([PDF](#)) oder auch Bilder.

Aufgrund der Vielzahl an angebotenen Dienstleistungen ist es schwierig herauszufinden, ob die angebotenen Dienstleistungen noch fehlerfrei arbeiten oder aus welchem Grund die Benutzer nicht mehr auf einen Dienst zugreifen können. Für diesen Zweck wurden Überwachungssysteme entwickelt, die den Status von verschiedenen Komponenten und den davon abhängigen Diensten überwachen und bei Veränderungen die Verantwortlichen darüber informieren.

Für einen möglichst störungsfreien Betrieb ist es notwendig, dass die Ergebnisse der Überwachung in periodischen Zeitabständen erneuert werden, damit ein auftretendes Problem schnellstmöglich erkannt und behoben werden



kann. Das Überwachungssystem soll so implementiert werden, dass Fehler erkannt werden, bevor die Nutzung der angebotenen Dienstleistungen davon beeinträchtigt wird. Dabei muss die zusätzliche Belastung des Netzwerkes und der überwachten Objekte durch die Überwachung eingeplant, die verwendete Netzwerkstruktur und die dadurch entstehende Abhängigkeit (von Netzwerknoten) beachtet und sicherheitstechnische Aspekte einer automatischen Überwachung bedacht werden.

Im Laufe dieser Arbeit soll eine Überwachung eines Dokumenten-Management-Systems unter Berücksichtigung der Funktions- und Arbeitsweise des eingesetzten Dokumenten-Management-Systems durch eine Open Source Überwachungsanwendung realisiert werden.

2 Aufgabenstellung

Um den Mitarbeitern des Karlsruhe Institute of Technology eine möglichst ausfallsichere Plattform für die zentrale Speicherung, Bearbeitung und Verwaltung von Dokumenten anbieten zu können, soll eine Überwachung implementiert werden. Diese Überwachung soll nicht nur die Anwendung, sondern auch den darunterliegenden Server bezüglich seiner Systemressourcen berücksichtigen. Dabei müssen Überwachungselemente gefunden werden, mit deren Überprüfung der eindeutige Zustand der Anwendung festgestellt und der störungsfreie Betrieb sichergestellt werden kann.

Für die Verwaltung von Webseiten, Dokumenten und Bildern wird das Dokumenten-Management-System **Oracle UCM** der Firma Oracle eingesetzt. Um die zu überwachenden Objekte zu ermitteln, ist das Verständnis über den Aufbau und der spezifischen Funktions- und Arbeitsweise des verwendeten Dokumenten-Management-Systems notwendig.

Als Überwachungssoftware wird die Open Source-Software Nagios verwendet. Zur Realisierung der Überwachung muss auf die interne Logik und auf die verschiedenen Methoden bezüglich der Ermittlung der Statusinformationen eingegangen werden. Dabei soll eine Übersicht über die unterschiedlichen Überwachungsmethoden von Nagios erstellt und unter Berücksichtigung des späteren Einsatzes bewertet werden. Mit den ausgewählten Methoden soll die Überwachung auf verschiedenen Ebenen realisiert werden.

Die Klassifizierung der Überwachungselemente ergibt sich aus der Gewichtung der einzelnen Elemente. Dabei soll die Anwendung auch reaktiv durch eine Auswertung von Logdateien auf Fehler überwacht werden.

Zur eindeutigen Erkennung von Fehlern, die während der Benutzung durch Anwender auftreten, sollen die typischen Aktionen der Benutzer simuliert werden. Für die Realisierung dieser Benutzersimulation muss die Anwendung über eine Schnittstelle verfügen, die sich durch ein Programm über das

Netzwerk ansprechen lässt. Dieses Programm soll die Benutzeraktionen automatisiert durchführen und der Überwachungssoftware Nagios die Ergebnisse der einzelnen Schritte übermitteln, damit der Fehlerzustand sofort erkannt und gleichzeitig seine Ursache eingegrenzt werden kann.

Dabei müssen bei der Programmentwicklung mögliche Konsequenzen aufgrund verschiedener Szenarien bedacht werden. Sollte die Anwendung bereits durch eine Vielzahl von Benutzern stark belastet sein, wird dadurch auch der Ablauf der Benutzersimulation verzögert. Eine solche Verzögerung soll von der Überwachungssoftware bzw. Benutzersimulation bei der Auswertung berücksichtigt werden.

Die Nutzung der Anwendung durch die eigentlichen Benutzer darf dabei nicht beeinträchtigt werden. Da die Ausführung der Benutzersimulation durch Nagios in kurzen Zeitabständen periodisch aufgerufen wird, müssen auch langfristige Auswirkungen wie das Überlaufen der Datenbank der Anwendung oder die Überfüllung des Festplattenspeichers des Anwendungsservers bedacht werden.

Für die Entwicklungsumgebung wird ein eigener Nagios-Server eingesetzt, deshalb muss die entwickelte Lösung auf den bereits vorhanden Nagios-Server exportierbar sein.

3 Grundlagen

In diesem Kapitel werden die Grundlagen von Überwachungssystemen und Dokumenten-Management-Systemen erläutert. Insbesondere wird auf Service-orientierte Architektur (**SOA**) und Web Services für die spätere Umsetzung eingegangen.

3.1 Überwachungssysteme

Überwachungssysteme wurden für den Zweck entwickelt den Status von verschiedenen Objekten meist über das Netzwerk zu überwachen und im Falle einer Statusänderung diese Information an die zugewiesenen Kontaktpersonen weiterleitet.

Generell unterscheidet man zwischen der Überwachung ermöglichten zu Grunde liegenden Hardware den so genannten Hosts und den auf diesen Hardwarerelementen aufsitzenden Diensten auch Services genannt.

Unter Hosts fallen nicht nur Server bzw. Computer, sondern auch Switches, Router oder auch dedizierte Überwachungshardware wie Sensoren für Temperatur, Luftfeuchtigkeit oder Rauchmelder. Die Services dieser Hosts weichen je nach Art der Hosts stark voneinander ab. Auf einem Server kann als Service ein Webserver im Betrieb sein, dessen Funktionalität sich über einen Aufruf einer Webseite überprüfen lässt. Bei einem Switch können beispielsweise als Service die Übertragungsrate, der Paketverlust oder der Portzustand überwacht werden.

Sehr wichtig ist bei einem Überwachungssystem die Gewichtung der erhaltenen Überwachungsinformationen.

Vor der Einführung eines Überwachungssystems muss sich mit den folgenden Punkten auseinandergesetzt werden.

3.1.1 Ressourcenbelastung

Die Einführung einer Überwachungssoftware bringt bei größeren Serverlandschaften eine nicht zu verachtende Netzwerk- und Prozessorbelastung mit sich. Dabei unterscheidet Josephsen die anfallende Belastung in zwei unterschiedliche Arten der Überwachung²:

Zentralisierte Überwachung Die Durchführung der Überprüfungen findet durch einen zentralen Überwachungsserver statt, der die Informationen über die einzelnen Hosts und Services über das Netzwerk abfragt. Diese Methode ist in der Regel vorzuziehen, da hierbei die zu überwachenden Geräte weniger belastet werden und die Konfiguration der einzelnen Kontrollschrifte zentral möglich ist.

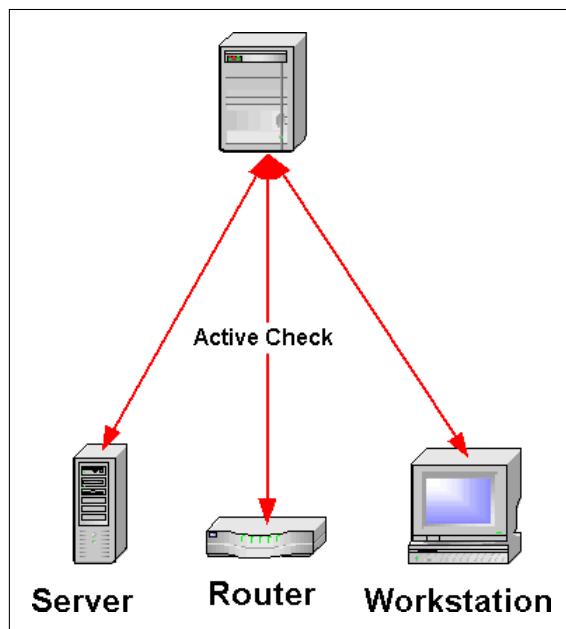


Abbildung 1: Zentralistische Bearbeitung

²Quelle: [Jose07] S. 4

Dezentralisierte Überwachung Bei einer sehr hohen Anzahl von zu überwachenden Objekten ist eine zentralisierte Ausführung nicht mehr von einem einzelnen Server tragbar. In diesem Fall ist das Überwachungssystem darauf angewiesen, dass die einzelnen Hosts die kontrollierenden Überprüfungen selbstständig durchführen und deren Ergebnisse an den Überwachungsserver weiterzuleiten.

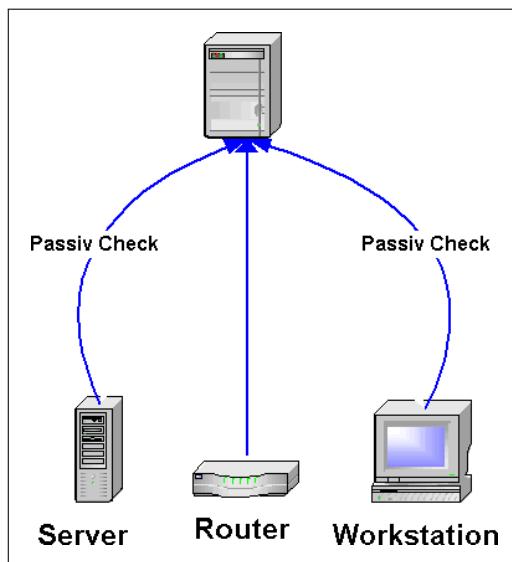


Abbildung 2: Ausgelagerte Bearbeitung

Um nicht komplett von einem Überwachungsserver abhängig zu sein, können weitere Überwachungsserver hinzugefügt werden. Diese können bei einem Ausfall des Hauptüberwachungsservers die Verantwortlichen informieren oder die zu überwachenden Objekte zur Lastenteilung untereinander aufteilen.

3.1.2 Netzwerkstruktur und Abhängigkeiten

Die Überwachung von Hosts und Services über das Netzwerk erzeugt normalerweise immer zusätzlichen IP-Traffic. Das bedeutet, dass jede Überquerung weiterer Netzwerkknoten, die zwischen dem Überwachungsserver und den zu überwachenden Geräten liegen, eine weitere Belastung für das Netzwerk bedeutet, sowie eine Abhängigkeit zwischen Host und Server einführt.

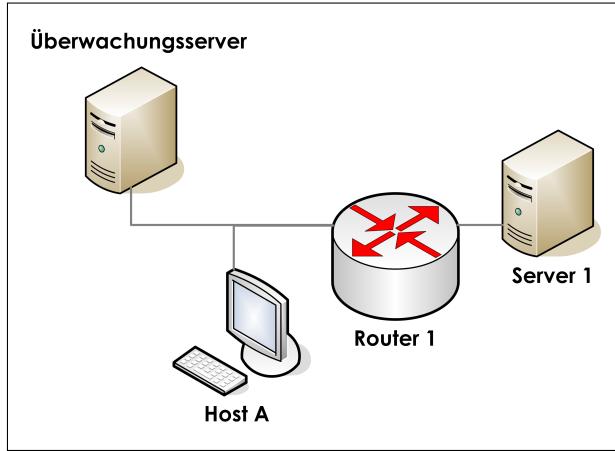


Abbildung 3: Zusätzliche Netzwerkabhängigkeit und Netzwerkbelastung

In der Abbildung 3 erzeugt der Router 1 die zuvor beschriebene zusätzliche Netzwerkabhängigkeit und Netzwerkbelastung, da der Server 1 bei einem Ausfall des Routers nicht mehr durch den Überwachungsserver erreichbar ist und jede Überprüfung, die vom Überwachungsserver gesendet wird den Router mit dem Routing der Pakete belastet.

Deshalb gilt es laut [Jose07] S. 5 folgende zwei Punkte beim Erstellen eines Überwachungssystems zu beachten:

Überwachungsredundanzen vermeiden Redundante Überwachung entsteht dadurch, dass der gleiche Service durch zwei Arten in unterschiedlicher Tiefe geprüft wird. Ein einfaches Beispiel ist die Überwachung eines Webservers auf dem Standardport 80. Eine Überwachungsmethode ist es diesen Port abzufragen und die entsprechende Rückantwort des Servers auszuwerten. Als zweiter Test soll die auf dem Webserver laufende Webseite überwacht werden. Dafür kann die jeweilige Webseite über die Adresse nach einem bestimmten Inhalt untersucht werden.

In beiden Fällen wird getestet, ob der Webserver über das Netzwerk ansprechbar ist, jedoch sagt der zweite Test zusätzlich noch aus, dass die korrekte Webseite angezeigt wird, somit wäre der erste Test überflüssig. Jedoch muss zuvor abgewogen werden, ob eine redundante Überwachung nicht so-

gar hilfreich bei der Ermittlung der Fehlerursache ist. So können beide Tests die Fehlerursache eingrenzen. Der erste Test überprüft, ob der Webserver erreichbar ist und der zweite Test kann erkennen, ob eine falsche bzw. veraltete Seite ausgeliefert wird.

Minimale Netzwerkbelastung Um bereits stark belastete Netzwerkpunkte zu entlasten, bietet es sich an, die Frequenz mit der die Test über das Netzwerk gesendet werden zu verringern. Die Aufstellung des Überwachungsservers ist dadurch gerade bei größeren Serverlandschaften sehr wichtig, da durch eine effiziente Platzierung womögliche Flaschenhälse in Form von veralteten Switches oder ähnlichem vermieden werden können.

3.1.3 Sicherheitsaspekte

Um erweiterte Statusinformationen über einen Prozess oder über die Arbeitsspeicherauslastung auszulesen, ist zusätzliche Software auf den Hosts nötig. Diese Software benötigt oft einen zusätzlichen geöffneten Port auf dem zu überwachendem Rechner, die einen neuen Angriffspunkt für Angreifer darstellen kann. Außerdem erhält der Überwachungsserver Ausführungsrechte auf dem Client, so dass eine weitere potentielle Sicherheitslücke in einem vermeintlich zuvor sicherem System entsteht. Jeder, der die Kontrolle über den Überwachungsserver besitzt oder sich als solcher ausgibt, kontrolliert somit gleichzeitig alle anderen überwachten Hosts.

Um dies zu verhindern gibt es verschiedene Ansätze. Als ersten Ansatz sollte der Port durch den Überwachungsserver mit dem Host kommuniziert vom Standardwert abweichen, damit nicht sofort erkennbar ist, dass sich eine angreifbare Überwachungssoftware auf dem Rechner befindet. Damit die über diesen Port versendeten Informationen nicht für Dritte zugänglich sind, bietet es sich an die auszutauschende Informationen mit einem Algorithmus zu verschlüsseln. Durch den Einsatz eines Verschlüsselungsalgorithmus

werden die Informationen nicht mehr im Klartext ausgetauscht, sondern Da die Möglichkeit einer Verschlüsselung der Datenübertragung nicht von jeder Überwachungssoftware angeboten wird, gilt diese Option als Auswahlkriterium in der späteren Umsetzung bzw. im produktivem Betrieb.

Des weiteren sollte die Erlaubnis der Abfrage der Überwachungsinformationen anhand der IP-Adresse eingeschränkt werden, so dass der Client nur Anfragen des Überwachungsservers akzeptiert. Durch diese Einschränkung kann vermieden werden, dass sensible Informationen aus den Antworten an unberechtigte Dritte übermittelt werden oder ein Denial of Service-Angriff (DoS) durch eine übermäßig hohe Anzahl an Anfragen an den Client gesendet wird, um eine Überlastung des Servers zu erreichen und diesen somit arbeitsunfähig zu machen.

3.2 Dokumenten-Management-Systeme

Um ein Dokumenten-Management-System (DMS) zu erläutern muss sich zuerst mit dem Begriff des „Dokuments“ auseinander gesetzt werden. In [DMS08] S. 2 wird ein Dokument durch folgende Punkte definiert:

- Ein Dokument fasst inhaltlich zusammengehörende Informationen strukturiert zusammen, die nicht ohne erheblichen Bedeutungsverlust weiter unterteilt werden können.
- Die Gesamtheit der Information ist für einen gewissen Zeitraum zu erhalten.
- Ein Dokument ist als Einheit ablegbar (speicherbar) und/oder versendbar und/oder wahrnehmbar (sehen, hören, fühlen).
- Das Dokument ist eigentlich der Träger, der die Informationen speichert, egal ob das Dokument ein Stück Paper, eine Datei auf einem Rechner, ein Videoband oder eine Tafel etc. ist. Dies bedeutet auch, dass es keine Bindung an Papier oder ein geschriebenes Wort gibt.

Des weiteren gibt es eine Differenzierung in zwei Definitionen:

„Als **Dokument im konventionellen Sinne** werden Dokumente bezeichnet, die als körperliches Dokumente (z. B. Papier) vorliegen, ursprünglich als körperliches Dokument vorlagen oder für die Publizierung auf einem körperlichen Medium vorgesehen sind.

Die Begrifflichkeit des **Dokuments im weiteren Sinne** erweitert den Begriff des Dokuments um semantisch zusammengehörende Informationsbestände, die für die Publikation in nicht-körperlichen Medien, z.B. Webseiten, Radio, Fernsehen o. ä. vorgesehen sind. Derartige Dokumente werden oft dynamisch gestaltet und zusammengestellt.“

[DMS08] S. 2

Dabei müssen auch Daten und Dokumente voneinander abgegrenzt werden. In [DMS08] S. 33 werden Daten im Allgemeinen als eher stark strukturierte Informationen gesehen, wobei Dokumente zumeist aus unstrukturierte bis zu schwach strukturierte Informationen bestehen. Eine eindeutige Klassifizierung eines vorhandenen Dokumentes ist jedoch nicht immer möglich, da sich oft Mischungen beider Klassen finden lassen. Ohne die dazugehörigen Metadaten besteht ein Bild aus unstrukturierten Informationen, daher auch **NCI-Dokument** für None-Coded Information genannt.

Die Einordnung, wann ein Dokument strukturierte oder unstrukturierte Informationen enthält, lässt an folgenden Beispielen verdeutlichen. Bei einem Bild oder Foto lassen sich die enthaltenen Informationen ohne zusätzliche Metadaten nicht eindeutig durch Computer bestimmen. Daher ist ein Bild, solange keine Metadaten darüber bekannt sind, ein eindeutiges Beispiel für **NCI-Dokumente** mit unstrukturierten Informationen. Im Gegensatz dazu lassen sich die Werte einer Tabelle oder eines Datensatzes durch die Spaltennamen eindeutig bestimmen und durch den Computer auslesen. Solche Daten

mit strukturierten Informationen werden daher auch als Dokumententyp mit Coded Information (**CI**) bezeichnet.

Der Anteil von strukturierten Informationen in einem Dokument nimmt von Bildern über Text zu Tabellen zu, da hier die Dokumente vollautomatisch auswertbar sind, siehe hierzu Abbildung 4.

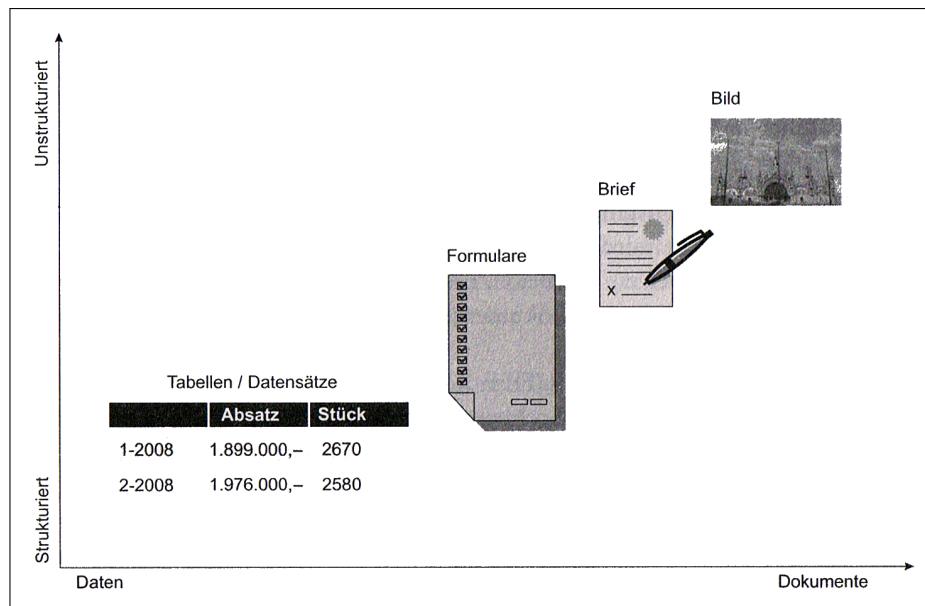


Abbildung 4: Anteil an strukturierten Informationen³

Unter **Dokumenten-Management** werden primär die Verwaltungsfunktionen Erfassung, Bearbeitung, Verwaltung und Speicherung von Dokumenten verstanden. [DMS08] S. 344.

Darunter fallen laut [DMS08] S. 3 folgende Punkte:

- Kennzeichnung und Beschreibung von Dokumenten (auch Metadaten des Dokuments genannt)
- Fortschreibung, Versionierung und Historienverwaltung von Dokumenten
- Ablage und Archivierung von Dokumenten

³Quelle: [DMS08] S. 33

- Verteilung und Umlauf von Dokumenten
- Suche nach Dokumenten bzw. Dokumenteninhalten
- Schutz der Dokumente vor Verfälschung, Missbrauch und Vernichtung
- Langfristiger Zugriff auf die Dokumente und Lesbarkeit der Dokumente
- Lebenslauf und Vernichtung von Dokumenten
- Regelung von Verantwortlichkeiten für Inhalt und Verwaltung von Dokumenten

Der Begriff „**Dokumenten-Management-System**“ muss auch in zwei verschiedene Sichtweisen differenziert werden:

„Bei **Dokumenten-Management-Systemen im engeren Sinne** geht es um die Logik der Verwaltung von Dokumenten, deren Status, Struktur, Lebenszyklus und Inhalt. Dokumente werden beschrieben, klassifiziert und in einer bestimmten logischen Struktur eingeordnet, damit sie einfach wieder gefunden werden können. Dokumente entstehen, werden verändert und (irgendwann) vernichtet.“

Den **Dokumenten-Management-Systemen im weiteren Sinne** ordnet man auch noch weitere Funktionalitäten zu, wie z. B. Schifterkennung, automatische Indizierung, [...], Publizierung. Hier lassen sich die Grenzen nicht mehr genau bestimmten!“

[DMS08] S. 5

Die Grundstruktur eines Dokumenten-Management-Systems kann man dadurch grob in folgender Abbildung zusammenfassen:

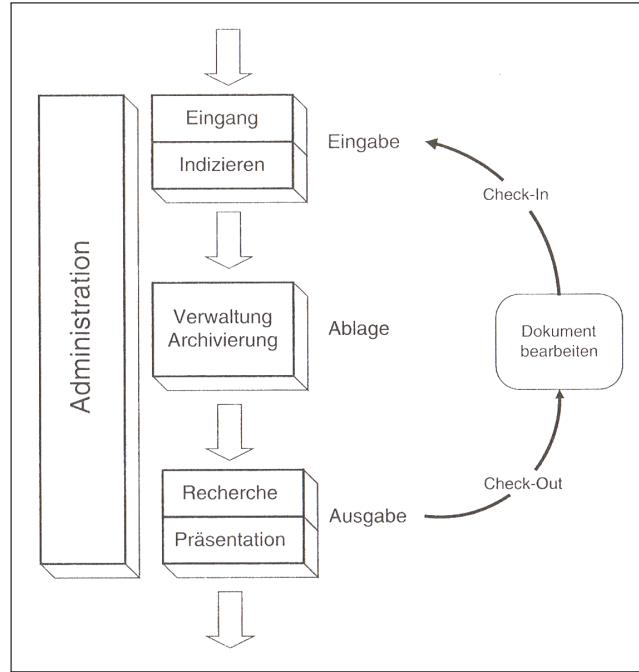


Abbildung 5: Aufgabenbereiche eines Dokumenten-Management-Systems⁴

Dabei wird ein **DMS**-System in drei verschiedene Teilbereiche aufgegliedert:

3.2.1 Eingabe

Unabhängig des Ursprungs oder der Art des Dokumentes besitzt der Funktionsbereich Eingabe die Aufgabe diese Dokumente dem Dokumenten-Management-System zuzuführen. Laut [DMS08] S. 40fff fallen in diesen Bereich zwei Funktionen:

Dokumenteneingang Hier wird das Zuspielen der Dokumente in das **DMS**-System durch verschiedene Methoden behandelt. Als mögliche Eingabe von Dokumenten kann sowohl das Einscannen von Textdokumente oder Bilder als auch der elektronische Eingang von Dokumenten durch E-Mail oder externen Anwendungen fungieren.

⁴Quelle: [DMS08] S. 38

Auch hier gilt zu unterscheiden, dass durch den Einstellvorgang erstellte Dokumente als **NCI**-Dokument abgelegt werden und bereits digitalisierte Dokumente sich zur Umwandlung zu **CI**-Dokumenten anbieten. Sobald der Inhalt von eingescannten Dokumenten zur weiteren Verarbeitung ausgelesen bzw. ausgewertet werden soll, müssen die Dokumente in ein **CI**-Format transformiert werden. Dies wird häufig durch eine **OCR**-Software realisiert, die beispielsweise das Bild eines eingescannten Briefes in Text umwandelt. Bereits im **CI**-Format vorliegende Dokumente müssen nicht transformiert werden, jedoch werden die Dokumente oft in anderen Formaten zusätzlich abgespeichert. Ein Beispiel ist die Umwandlung eines Microsoft Word-Dokumentes in ein **PDF**-Dokument oder von einem RAW-Bild in das verbreitete JPG-Format.

Indizierung Bei der Indizierung werden Dokumente zur eindeutigen Identifikation mit Attributen versehen. Diese Attribute werden teilweise automatisch durch das **DMS**-System anhand einer hochzählenden Identifikationsnummer oder manuell durch den Benutzer beim Einstellen des Dokumentes hinzugefügt. Solche Attribute werden auch als Metadaten des Dokumentes bezeichnet und meist als zusätzliche Suchkriterien angeboten.

Dabei werden in [DMS08] S. 44 zwei verschiedene Methoden zur automatischen Klassifizierung genannt. Beim wissensbasiertem Ansatz wird mittels umfangreichem Wissen über das Umfeld der Dokumente und dadurch abgeleitete Regeln dem System ermöglicht diese Dokumente automatisch einzurichten und zu indizieren. Eine weitere Möglichkeit eröffnet sich durch das Verwenden von neuronalen Netzen. Hierbei wird durch die Vorarbeit eines Menschen Beispiele geschaffen anhand welcher sich das System selbstständig Auswahlkriterien erzeugt. Je mehr korrekte Beispiele vorgegeben werden, desto besser und zuverlässiger arbeitet die automatische Klassifizierung.

3.2.2 Verwaltung und Archivierung

Bei der **Verwaltung** werden die Probleme beim *Check-in* (Einspielen des Dokumentes), Bearbeitung und *Check-out* (Signalisieren der Weiterbearbeitung) behandelt, siehe auch Abbildung 5. Wie auch bei einer Datenbank müssen Dokumente, die gerade bearbeitet werden, für andere Benutzer für Änderungen gesperrt werden, damit keine Inkonsistenzen auftreten können. Nach einer Bearbeitung und dem Check-in des abgeänderten Dokumentes muss die Versionsverwaltung des DMS-Systems beide Versionen beibehalten und die ursprüngliche Version als veraltet und die neue Version als solche kennzeichnen. Zusätzlich muss die Wiederherstellung einer älteren Revision als aktuelles Dokument unterstützt werden.

Die **Archivierung** befasst sich mit der Sicherung und Wiederherstellung von Dokumenten und deren Metadaten. Im Zusammenhang mit DMS-Systemen spielt auch eine revisionssichere Archivierung oft eine Rolle. Dabei müssen laut [DMS08] S. 288 unter anderem bestimmte Punkte eingehalten werden:

- Jedes Dokument muss unveränderbar archiviert werden.
- Es darf kein Dokument auf dem Weg ins Archiv oder im Archiv selbst verloren gehen.
- Kein Dokument darf während seiner vorgesehenen Lebenszeit zerstört werden können.
- Jedes Dokument muss in genau der gleichen Form, wie es erfasst wurde, wieder angezeigt und gedruckt werden können.

3.2.3 Ausgabe

Wie die Eingabe besteht die Ausgabe aus zwei Funktionen:

Recherche Die Recherche ist die Suche nach einem Dokument entweder durch eine strukturierte Suche anhand von zuvor eingetragenen Attributen

(Autor, Erstellungsdatum, Speichergröße usw.) oder durch eine Volltextsuche.

Die **strukturierte Suche** ist nur bei einer qualitativ hochwertigen Indizierung effizient, bietet dafür auch mit guter zeitlichen Performanz die besten Ergebnisse, sofern die Indizierung entsprechend aufgebaut wurde.

Die **Volltextsuche** besteht aus einer ordinären Suche durch den Inhalt der Dokumente nach den eingegebenen Suchbegriffen. Daher ist die Qualität der Suchergebnisse unabhängig von der Qualität der Indizierung. Jedoch können nur **CI**-Dokumente, deren Informationen auch durch den Computer auslesbar und interpretierbar sind, durchsucht werden. **NCI**-Dokumente wie Bilder oder Videos können ohne Metadaten durch die Volltextsuche nicht recherchiert werden.

Reproduktion In diesem Teilbereich können die gespeicherten Dokumente wieder vom Benutzer abgerufen werden. Dies ist durch eine einfache Anzeige im Webbrowser, eine Weiterleitung per E-Mail oder eine Sendung als Druckauftrag möglich.

3.3 Content-Management-Systeme

Bei einem Content-Management-System (**CMS**) steht nicht mehr das eigentliche Dokument im Vordergrund, sondern vielmehr der enthaltene Informationsgehalt des Dokuments. Der Unterschied zwischen einem **DMS** und einem **CMS** besteht laut [DMS08] S. 114 im Folgenden:

„Ein **DMS** hat als kleinstes Objekt der Betrachtung eines einzelnen Dokument. [...] Content-Management ist auf logische Informationseinheiten ausgerichtet. Es ist z.B. das Ziel des Content-Managements, Inhalte, die auf mehrere Quellen verteilt sind, neue zusammenzustellen und daraus z.B. ein neues Dokument zu generieren.“

[DMS08] S. 114f

Die folgende Abbildung soll den charakteristischen Unterschied zwischen **CMS**-Systemen und **DMS**-Systemen verdeutlichen.

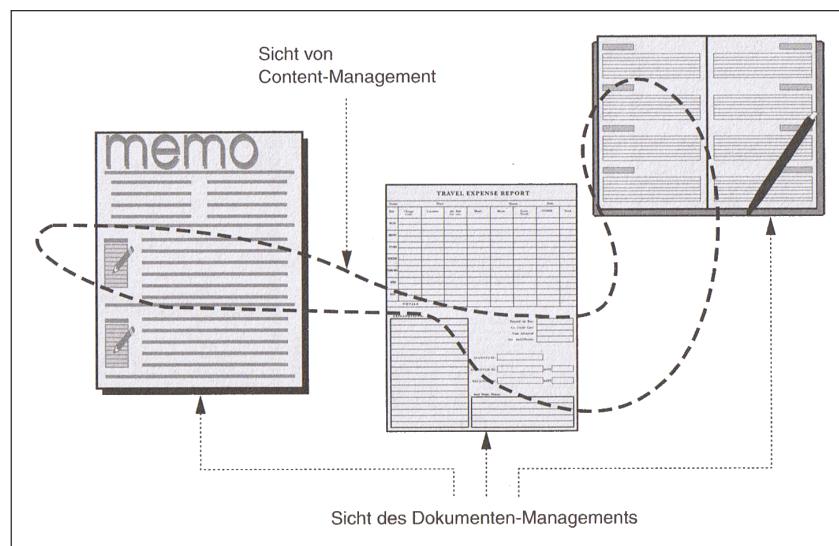


Abbildung 6: Sichtweise CMS gegenüber DMS⁵

⁵Quelle: [DMS08] S. 115

Wie zuvor beschrieben ist die Sichtweise eines **DMS** nur auf die einzelnen Dokumente beschränkt, während ein **CMS** einzelne Informationsbausteine aus den Dokumenten extrahieren und ggf. zu einem neuen Dokument verschmelzen kann. Die Sichtweise des **CMS** wird durch das gestrichelte Polygon dargestellt, welches hier dokumentenübergreifend abgebildet ist.

Der tiefere Sinn eines **CMS**-Systems ist laut Oracle folgendermaßen definiert:

„The key to a successful content management implementation is unlocking the value of content by making it as easy as possible for it to be consumed. This means that any piece of content must be available to any consumer, no matter what their method of access.“

[UCM07] S. 12

Ein **CMS** soll die Informationen jedwedem Inhalts extrahieren und jedes Element dieser Information den Benutzern zugänglich machen, unabhängig von der Art des Zugriffs. Dieses Konzept soll in Abbildung 7 verdeutlicht werden.

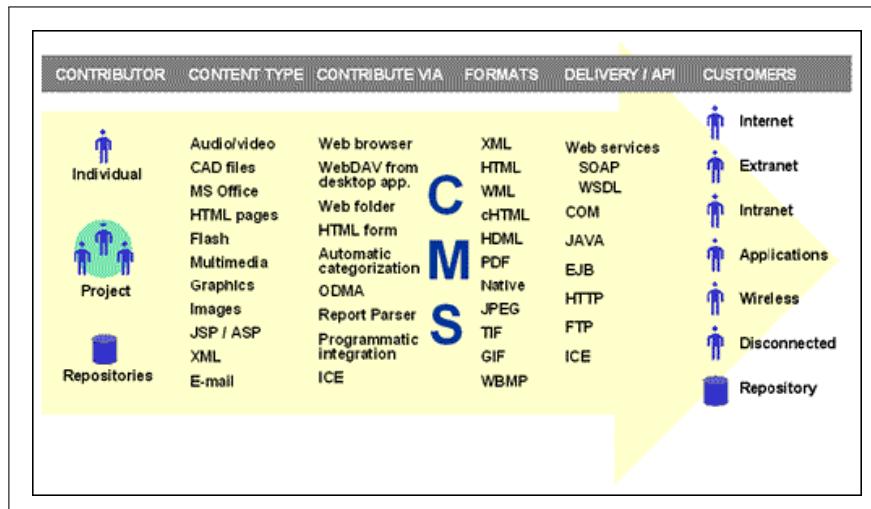


Abbildung 7: „any-to-any“ Content-Management Konzept⁶

⁶Quelle: [UCM07] S. 12

Das **CMS** steht hier in der Mitte der Abbildung als Medium zwischen den verschiedenen Inhalten, eingestellt von den *Contributors* (links), und den Anwendern, die auf transformierte Versionen der Inhalte durch unterschiedliche Arten zugreifen (rechts).

3.4 Enterprise-Content-Management-Systeme

Der Begriff Enterprise-Content-Management (**ECM**) wird auch häufig in Verbindung mit **CMS**-Systemen und **DMS**-Systemen genannt. Laut der „Association for Information and Image Management“ (**AIIM**⁷) umfasst dieser Begriff die Verwaltungsfunktionen von Unternehmensinformationen in unterschiedlichen Dokumentformaten.⁸ Diese Funktionen werden laut [DMS08] S. 116 durch verschiedene „Systeme wie Dokumenten-Management, Groupware, Workflow, Input- und Output-Management, (Web-)Content-management, Archivierung, Records-Management und andere“ bereitgestellt.

3.5 Serviceorientierte Architektur

Eine eindeutige und einheitliche Definition einer Serviceorientierter Architektur (**SOA**) existiert nicht. Einen Versuch einer Definition wird in [SOA07] beschrieben:

„[...] a service oriented architecture is an architecture for building business applications as a set of loosely coupled black-box components orchestrated to deliver a well-defined level of service by linking together business processes.“

[SOA07] S. 27

⁷Die AIIM ist eine Gesellschaft von internationalen Herstellern und Anwendern von Informations- und Dokumenten-Mangement-Systemen

⁸Quelle: <http://www.aiim.org/What-is-ECM-Enterprise-Content-Management.aspx>

SOA ist ein Ansatz im Bereich der Informationstechnik um Anwendungen oder einzelne Dienste aus verschiedenen Geschäftsprozessen zu bilden.

Melzer bietet eine ausführlichere Definition:

„Unter einer **SOA** versteht man eine Systemarchitektur, die vielfältige, verschiedene und eventuell inkompatible Methoden oder Applikationen als wiederverwendbare und offen zugreifbare Dienste repräsentiert und dadurch eine plattform- und sprachenunabhängige Nutzung und Wiederverwendung ermöglicht.“

[Melzer08] S. 13

Zur Verdeutlichung einer **SOA** kann ein beispielhafter und vereinfachter Aufbau eines Online-Shops verwendet werden.

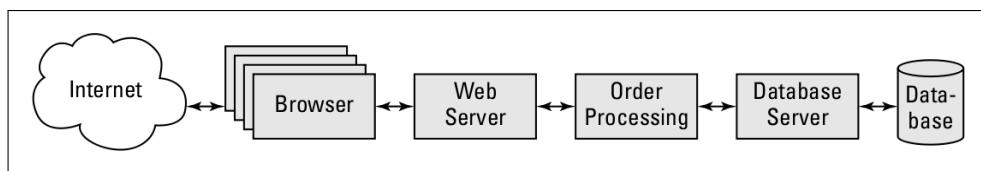


Abbildung 8: Simple Software Architektur eines Webshops⁹

Durch den gewöhnlichen Browser können Benutzer auf die Webseite des Webservers zugreifen um dort auf die eigentliche Anwendung des Webshops *Order Processing* zuzugreifen. Dabei werden durch einen Datenbankserver die Informationen in einer Datenbank gespeichert oder von dort der Webshop-Anwendung zugänglich gemacht. Welche Funktion die Anwendung *Order Processing* ausführt hängt von den Aufforderungen des Benutzers durch den Browser ab.

⁹Quelle: [SOA07] S. 18

Dieser Struktur wird nun ein Serviceorientierte Komponente *Credit Checking* hinzugefügt, siehe Abbildung 9.

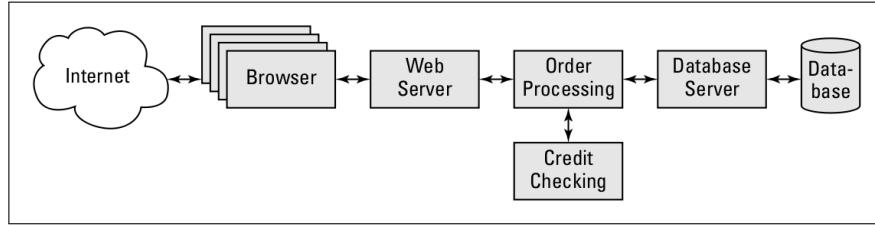


Abbildung 9: Hinzugefügte Serviceorientierte Komponente¹⁰

Dabei hat die eigentliche Anwendung des Webshops keine Kenntnis wie die Komponente *Credit Checking* intern abläuft, sondern übergibt nur die essentiellen Informationen, in diesem Fall die Kreditkartendaten, an die Komponente. Für die Anwendung ist irrelevant, ob diese Komponenten eine externe Datenbank oder Webseite nach der Kreditwürdigkeit des Benutzers befragen, solange die Komponente auswertbare Informationen (zahlungsfähig ja/nein) an die Webshop-Anwendung liefert. Für die Anwendung *Order Processing* ist die Komponente *Credit Checking* eine so genannte **black box**. Die komplexen Berechnungen und Algorithmen zur Bestimmung der Kreditwürdigkeit des Benutzers werden komplett verdeckt, so dass nur die Kreditkarteninformationen der Komponente zu übergeben sind. Die Komponente *Credit Checking* steht der Webshop Anwendung als **abstrahierter Dienst bzw. Service** zur Verfügung.

3.6 Web Services-Architektur

Wie bei dem Begriff **SOA** gibt es für Web Services keine allgemein gültige Definition, jedoch überlappen sich Definitionsvorschläge in verschiedenen Gesichtspunkten. Laut Melzer ([Melzer08] S. 55) bietet das World Wide Web Consortium (**W3C**) den konkretesten Ansatz einer passenden Definition.

¹⁰Quelle: [SOA07] S. 20

„A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically **WSDL**). Other systems interact with the Web service in a manner prescribed by its description using **SOAP** messages, typically conveyed using **HTTP** with an **XML** serialization in conjunction with other Web-related standards.“

[W3WS04] S. 7

Ein Web Service ist so aufgebaut, dass ein Zusammenspiel zwischen Rechner über ein Netzwerk möglich ist. Dabei ist Schnittstelle des Web Services in einem maschinell interpretierbaren Format gehalten, so dass andere Systeme auf diese Schnittstelle zugreifen können. Dieser Zugriff findet durch das Simple Object Access Protocol (**SOAP**) statt, welches üblicherweise über das Hypertext Transfer Protocol (**HTTP**) versendet wird. Die **SOAP**-Nachrichten sind nach dem **XML**-Schema zusammen mit anderen Web-Standards aufgebaut. Dadurch können die Nachrichten von beiden Seiten (Client und Server) interpretiert werden.

Daraus leitet Melzer folgende Spezifikationen für eine Web Services-Architektur ab:

SOAP beschreibt das **XML**-basierte Nachrichtenformat der Kommunikation und dessen Einbettung in ein Transportprotokoll.

WSDL ist eine - ebenfalls **XML**-basierte - Beschreibungssprache, um Web Services (Dienste) zu beschreiben.

UDDI beschreibt einen Verzeichnisdienst für Web Services. **UDDI** (Universal Description, Discovery and Integration protocol) spezifiziert eine standardisierte Verzeichnisstruktur für die Verwaltung von Web Services-Metadaten.

Zu den Metadaten zählen allgemeine Anforderungen, Web Services-Eigenschaften oder die benötigten Informationen zum Auffinden von Web Services.

[Melzer08] S. 55

Als Beispiel für eine **SOAP**-Kommunikation verschickt ein Client zwei Zahlenwerte, die vom Server addiert werden sollen:

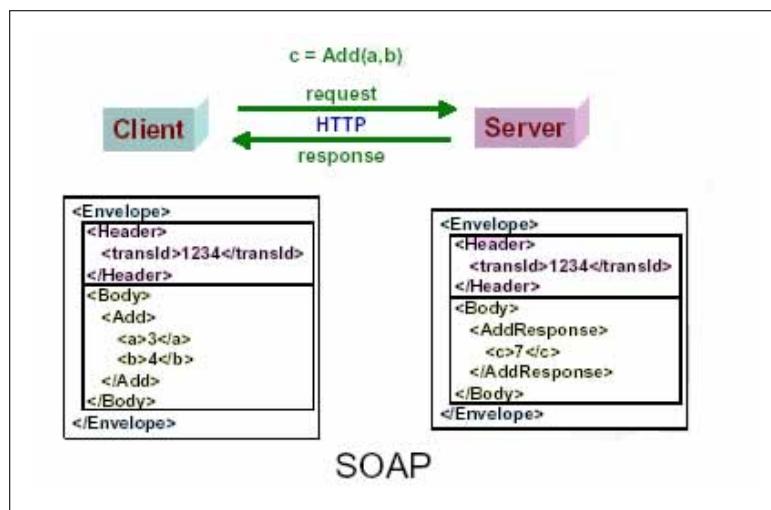


Abbildung 10: Kommunikationprotokoll **SOAP**¹¹

Der Server entpackt die **SOAP**-Nachricht und führt mit den zwei Zahlenwerten die Addition aus. Das Ergebnis der Rechnung wird im Anschluss wieder als Nachricht im **SOAP**-Format an den Client zurückgesendet.

Der Ablauf der Benutzung eines Web Services soll durch Abbildung 11 verdeutlicht werden.

1. Der Anbieter des Web Services muss seinen Dienst durch eine **WSDL**-Datei in Form einer **XML**-Datei dem Dienstverzeichnis bekannt geben.
2. Erst dann können mögliche Nutzer dieses Dienstes den Web Service im **UDDI**-basiertem Dienstverzeichnis finden. Die Suchanfrage findet über eine **SOAP**-Schnittstelle statt.

¹¹Quelle: [muSOAP]

3. Ein Verweis auf den Dienst in Form einer **WSDL**-Datei wird an den Dienstbenutzer als Antwort der Suchanfrage gesendet.
4. Durch diesen Verweis erfährt der Benutzer die Adresse des Dienstangebieters und kann die Beschreibung des Web Services abfragen.
5. Nach Erhalt dieser Beschreibung kann der eigentliche Webdienst mittels **SOAP** verwendet werden.

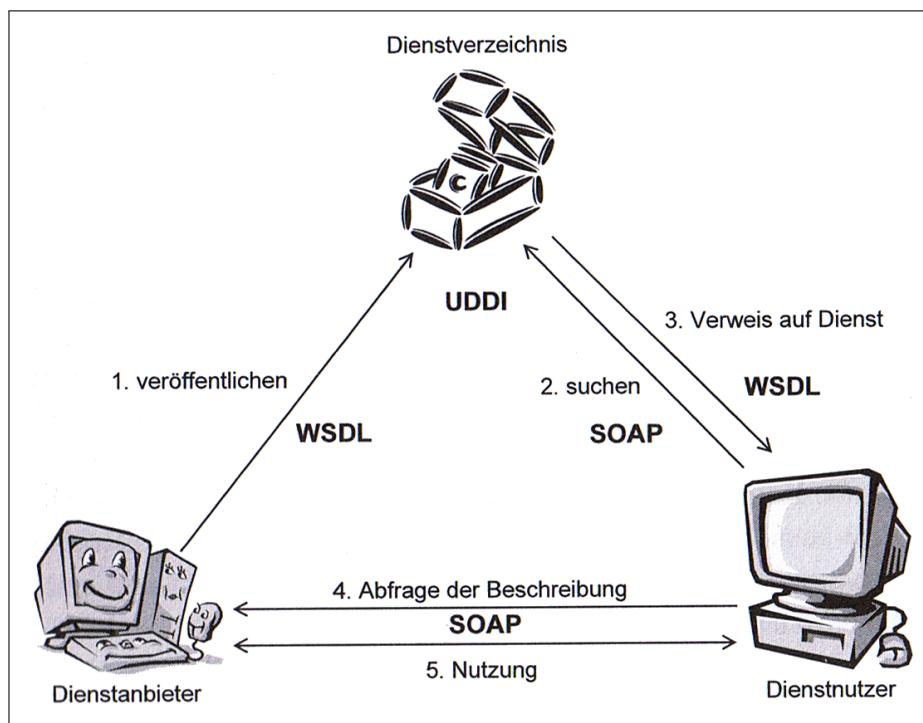


Abbildung 11: Ablauf einer Web Service-Benutzung¹²

Dabei erwähnt Metzer, dass ein Verzeichnisdienst keine Notwendigkeit für die Verwendung eines Web Services ist¹³ und, dass die Verwendung von **UDDI** in Firmen immer mehr nachlässt¹⁴.

¹²Quelle: [Melzer08] S. 56

¹³Quelle: [Melzer08] S. 56

¹⁴Quelle: [Melzer08] S. 158

4 Nagios

In diesem Kapitel wird der Aufbau von Nagios erläutert und ein Überblick über die unterschiedlichen Überwachungsmethoden gegeben.

4.1 Allgemein

Nagios dient zum Überwachen von Hosts und deren Services in komplexen Infrastrukturen und wurde von dem Amerikaner Ethan Galstad seit 1999¹⁵ - damals unter der Vorgängerversion NetSaint - entwickelt und bis heute gepflegt.

Der Name Nagios ist ein Akronym für „Nagios Ain’t Gonna Insist On Sainthood“, dabei ist der Begriff Sainthood eine Anspielung auf die Vorgängerversion Net-Saint.¹⁶

Galstad gründete aufgrund der vielfältigen und positiven Resonanz am 9. November 2007 die „Nagios Enterprises LLC“, welche Nagios als kommerzielle Dienstleistung anbietet. Die Software selbst blieb weiterhin unter der freien Lizenz „GNU General Public License version 2“¹⁷ verfügbar. Diese erlaubt Einblick in den Programmcode und das Modifizieren der Anwendung nach eigenen Vorstellungen.

Nagios erfreut sich hoher Beliebtheit aufgrund der großen Community, die Tipps, Ratschläge und auch eigene Nagios-Plugins kostenlos anbietet. Außerdem können selbst mit geringen Programmierkenntnissen zusätzliche Skripte zur Überwachung geschrieben werden, wenn ein spezieller Anwendungsfall dies erfordert.

Nagios benötigt eine Unix-ähnliche Plattform und kann nicht unter Windows-Betriebssystemen betrieben werden.

¹⁵Quelle: <http://www.netsaint.org/changelog.php>

¹⁶Quelle: [NagiosFAQ]

¹⁷Quelle: <http://www.gnu.org/licenses/old-licenses/gpl-2.0.txt>

4.2 Aufbau

Barth schreibt über Nagios:

„Die große Stärke von Nagios - auch im Vergleich zu anderen Netzwerküberwachungstools - liegt in seinem modularen Aufbau: Der Nagios-Kern enthält keinen einzigen Test, stattdessen verwendet er für Service- und Host-Checks externe Programme, die als *Plugins* bezeichnet werden.“

[Barth08] S. 25

Dieser „Kern“ beinhaltet das komplette Benachrichtigungssystem mit Kontaktadressen und Benachrichtigungsvorgaben (Zeit, Art, zusätzliche Kriterien), die Hosts- und Servicedefinitionen inklusive deren Gruppierungen und schließlich das Webinterface.

Die eigentlichen Checks in Form der selbständigen Plugins sind abgekapselt von diesem Kern, siehe Abbildung 12.

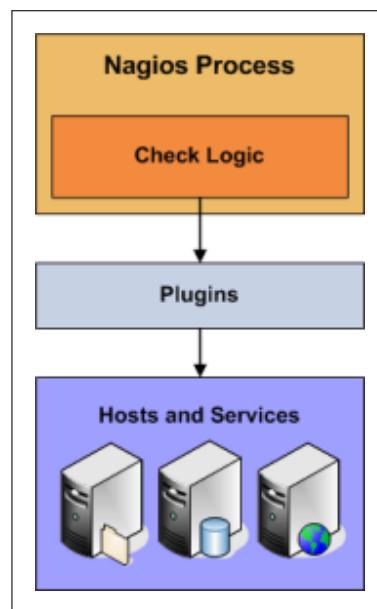


Abbildung 12: Plugins als separate Komponente¹⁸

¹⁸Quelle: http://nagios.sourceforge.net/docs/3_0/images/activechecks.png

Damit Nagios die gewünschten Server überwachen kann, müssen sie der Anwendung zuerst bekannt gemacht werden. Dies wird über das Anlegen einer Konfigurationsdatei mit einem Host-Objekt erreicht. Dabei richtet sich die Definition des Host-Objektes nach dem Schema, welches für alle Objektdefinitionen (Services, Kontakt, Gruppen, Kommandos etc.) gilt:

```
1 define object-type {
2     parameter value
3     parameter value ...
4 }
```

Listing 1: Nagiosschema für Objektdefinitionen

Eine gültige Host-Definition muss mindestens folgende Elemente besitzen:

```
1 define host{
2     host_name          example.kit.edu #Referenzname des Servers
3     alias              Oracle UCM Server #Weitere Bezeichnung
4     address            example.kit.edu #FQDN des Rechners
5     max_check_attempts 4 #Anzahl der Checks zum Wechsel von Soft-
6                   zu Hard-State
7     check_period       24x7 #Zeitraum der aktiven Checks
8     contact_groups    UCM-admins #Zu alarmierende Benutzergruppe
9     notification_interval 120 #Minuten bis Alarmierung wiederholt wird
10    notification_period 24x7 #Zeitraum der Benachrichtigungen
11 }
```

Listing 2: Definition eines Hostobjektes

In der Praxis werden öfters verwendete Attribute wie die Kontaktgruppe oder der Zeitraum für die aktiven Checks durch Verwendung eines übergeordneten Host-Objektes nach unten vererbt. Dadurch müssen nur noch die spezifischen Informationen des Servers eingetragen werden.

```
1 define host {
2     use           windows-server #Oberklasse dieses Host-Objektes
3     host_name    example.kit.edu
4     alias        Oracle UCM Server
5     address      example.kit.edu }
```

Listing 3: Verkürzte Definition eines Hostobjektes

Mit dieser Hostdefinition wird der Rechner im Webinterface von Nagios bereits angezeigt:

Host ↑\v	Service ↑\v	Status ↑\v	Last Check ↑\v	Duration ↑\v	Attempt ↑\v	Status Information
example.kit.edu	PING	OK	2009-07-23 13:47:24	0d 0h 3m 15s	1/4	PING OK - Packet loss = 0%, RTA = 26.66 ms

Abbildung 13: Anzeige des Servers im Webinterface von Nagios

Damit wird die Erreichbarkeit über das Netzwerk mit einem Ping getestet. Um weitere Informationen zu erhalten, müssen die gewünschten Plugins explizit aus dem Nagios-Repertoire dem zu überwachenden Computer mit einem ähnlichen Schema zugewiesen werden. Eine beispielhafte Servicedefinition für die Überwachung des Webservers auf dem Host *example.kit.edu* wird in CodeListing 5 gezeigt.

```

1 define service{
2     use          generic-service #Oberklasse dieses Service-Objektes
3     host_name    example.kit.edu
4     service_description HTTP Server #Bezeichnung des Checks
5     check_command   check_http #Angabe des Nagios-Plugins
6 }
```

Listing 4: Verkürzte Definition eines Hostobjektes

Die Plugins werden durch die Servicedefinitionen mit den jeweiligen Hosts verbunden und durch das Attribut *check_command* mit ggf. veränderten Parametern durch Nagios aufgerufen. Nagios wird in einem modifizierbaren Zeitintervall alle vom Benutzer definierten Host- und Servicechecks überprüfen und die Ergebnisse der entsprechenden Plugins auswerten.

Nagios-Plugins und Ausgabe Weiterhin beschreibt Barth die Plugins folgendermaßen:

„Jedes Plugin, das bei Host- und Service-Checks zum Einsatz kommt, ist ein eigenes, selbständiges Programm, das sich auch unabhängig von Nagios benutzen lässt.“

[Barth08] S. 105

Daher lassen sich die Parameter eines Plugins in der Kommandozeile überprüfen:

```
paul@iwrpaul:/usr/lib/nagios/plugins$ ./check_swap -w 20 -c 10
SWAP OK - 96% free (1826 MB out of 1906 MB) |swap=1826MB;0;0;0;1906
```

Abbildung 14: Beispielhafte manuelle Ausführung eines Servicechecks

Die Ausgabe des Plugins gibt den Zustand des Services an. In diesem Fall wird kein Schwellwert überschritten, daher die Meldung „*SWAP OK*“. Dieses Plugin liefert noch zusätzliche Performance-Informationen, die mit externen Programmen ausgewertet, gespeichert und visualisiert werden können. Standardmäßig werden die Performanzdaten von der normalen Ausgabe mit einem „|“ getrennt. Jedoch können auch Werte aus der normalen Textausgabe für die Visualisierung verwendet werden, so dass in diesem Beispiel keine Berechnung des Prozentsatzes notwendig wäre.

Um den Service mit den angegebenen Schwellwerten in Abbildung 14 von Nagios überwachen zu lassen, muss folgende Servicedefinition in die Konfigurationsdatei eingetragen werden:

```

1 #Test des Swap-Speichers mit WARNING und CRITICAL Schwellwertparameter
2 define service{
3   use generic-service
4   host_name     example.kit.edu
5   service_description Swap Disk Space
6   check_command check_swap!-w 20% -c 10%
7 }
```

Listing 5: Beispielhafte Definition eines Servicechecks

Die Plugins liefern dabei verschiedene Rückgabewerte:

Status	Bezeichnung	Beschreibung
0	OK	Alles in Ordnung
1	WARNING	Die Warnschwelle wurde überschritten, die kritische Schwelle ist aber noch nicht erreicht.
2	CRITICAL	Entweder wurde die kritische Schwelle überschritten oder das Plugin hat den Test nach einem Timeout abgebrochen.
3	UNKNOWN	Innerhalb des Plugins trat ein Fehler auf (zum Beispiel weil falsche Parameter verwendet wurden)

Tabelle 1: Rückgabewerte für Nagios-Plugins¹⁹

¹⁹Quelle: [Barth08] S. 105f

Anhand dieser Werte wertet Nagios gezielt den Status des jeweiligen Objektes (Host oder Service) aus.

Hard und Soft States Weiterhin gibt es weiche („*Soft States*“) und harte Zustände („*Hard States*“):

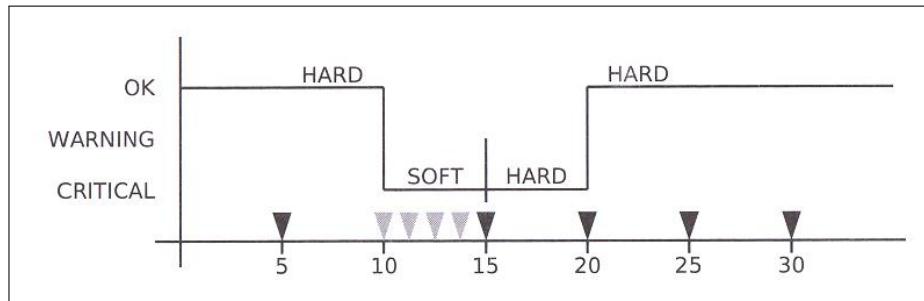


Abbildung 15: Beispiel für den zeitlichen Verlauf durch vers. Zustände²⁰

Ausgehend von einem „OK“-Zustand wird in diesem Beispiel jede fünf Minuten periodisch überprüft, ob sich der Status des überwachten Objektes verändert hat. Nach zehn Minuten wird eine Änderung des Zustandes durch das jeweilige Plugin gemeldet. Hier im Beispiel wechselt der Zustand nach „CRITICAL“, zunächst allerdings als Soft State. Daher wird durch Nagios noch keine Benachrichtigung versendet, da es sich um eine Falschmeldung, auch *False Positive* genannt, handeln kann. Aufgrund einer kurzfristigen hohen Auslastung des Netzwerkes oder um ein kurzzeitiges Problem, welches sich von alleine wieder normalisiert wie bspw. die Prozessorauslastung nach Beenden einer Anwendung.

Um False Positive-Meldungen zu verhindern, wird der im Soft State befindliche Service bzw. Host mit einer höheren Frequenz überprüft. Sollten diese Überprüfungen den vorherigen Zustand bestätigen, verfestigt sich der aktuelle Zustand, man spricht nun von einem Hard State. Erst in diesem Moment werden die entsprechenden Kontaktpersonen über den kritischen Zustand benachrichtigt. Sollte sich der Zustand wieder in den Normalzustand begeben

²⁰Quelle: [Barth08] S. 95

und dieser Zustandsübergang wird von dem Plugin festgestellt, wird dies an den Nagios-Server gemeldet. Ein Übergang zu dem „OK“-Status wird sofort als Hard State festgesetzt und führt zur sofortigen Benachrichtigung durch Nagios.

Flapping Nagios besitzt eine spezielle Funktion um sich zu schnell ändernde Zustände automatisch zu erkennen und die Benachrichtigung von diesen Objekten zu unterbinden. Das Verhalten dieser schnell wechselnde Zustände wird auch als *Flapping* bezeichnet und deren Erkennung durch Nagios als „*Flap Detection*“. Bei dieser Flap Detection speichert Nagios die letzten 20 Zustände und errechnet durch einen Algorithmus, welcher die aktuelleren Zuständen höher gewichtet, eine prozentuale Zustandsänderung.

Im folgenden Beispiel wurden sieben Zustandswechsel erfasst.

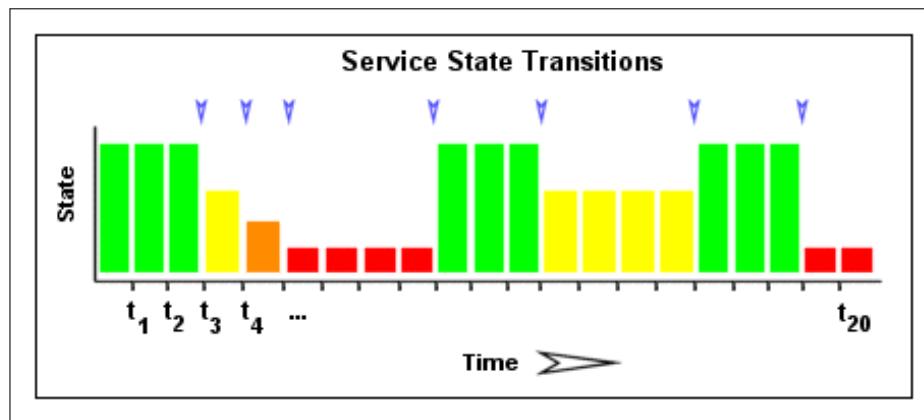


Abbildung 16: Verlauf von sich schnell wechselnden Zuständen²¹

Dadurch ergibt sich ein Wechselzustand von:

$$\frac{7}{20} = 0,35 = 35\%$$

Standardmäßig deklariert Nagios einen Host oder Service ab 25% als Flapping und unterbindet die Benachrichtigung.

²¹Quelle: http://nagios.sourceforge.net/docs/3_0/images/statetransitions.png

4.3 Überprüfungsmethoden

Man unterscheidet generell zwischen aktiven und passiven Checks.

4.3.1 Aktive Checks

Aktive Checks werden vom Nagios-Server direkt ausgeführt und holen die Informationen über die Zustände auf verschiedene Art und Weise ein. Nagios erwartet nach einem bestimmten Zeitintervall neue aktualisierte Informationen und gibt eine Alarmmeldung aus, wenn keine neuen Informationen angekommen sind.

4.3.2 Passive Checks

Bei passiven Checks werden die Scripts und Programme, die die Ergebnisse der zu überwachenden Objekte sammeln, selbstständig von dem zu überwachendem Computer ausgeführt. Der Nagios-Server nimmt die Ergebnisse von diesen Checks nur entgegen und führt sich nicht selbst aus. Da Nagios somit keine Kontrolle über die Ausführung der Plugins hat, können die Ergebnisse auch asynchron zu den anderen Plugins eintreffen.

Nagios bietet verschiedene Möglichkeiten an Informationen einzuholen:

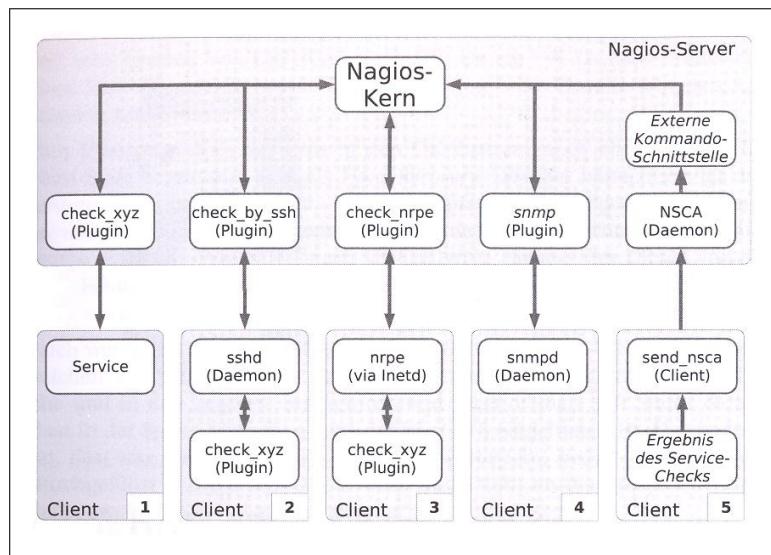


Abbildung 17: Verschiedene Überwachungsmöglichkeiten von Nagios²²

Methode 1 - Netzwerkdienste Dienste, die über das Netzwerk ansprechbar sind, wie bei einem Web- oder **FTP**-Server, lassen sich direkt über das Netz auf ihren Zustand überprüfen. Hierfür muss dem entsprechendem Plugin nur die Netzwerkadresse mitgeteilt werden, siehe Abbildung 18 als beispielhafte Überprüfung eines Webservers.

```
root@nagiosdev:/usr/lib/nagios/plugins# ./check_http -H example.kit.edu
HTTP OK HTTP/1.1 200 OK - 684 bytes in 0.001 seconds |time=0.001317s;;;0.000000 size=684B;;0
```

Abbildung 18: Ausführung eines netzwerkbasierenden Servicechecks

Der zuvor gezeigte Test eines netzwerkbasierenden Dienstes wird in Abbildung 17 mit dem Client-Rechner 1 abgebildet. Dies ist die einfachste Überwachungsmethode, da keine zusätzlichen Programme oder aufwändige Konfiguration benötigt wird. Vorteilhaft ist auch, dass der Dienst über das Netzwerk getestet wird, so wie der Benutzer auch auf den Dienst zugreift. Damit können auch gleichzeitig andere Knotenpunkte wie Switches überwacht werden.

Methode 2 - SSH Falls es sich beim Client um einen auf Unix basierenden Server handelt, ist meistens der Zugriff per **SSH** verfügbar. Dazu muss auf dem Client ein **SSH**-Benutzerkonto für Nagios angelegt sein und die öffentlichen Schlüssel auf dem Host abgelegt werden, damit keine passwortabhängige Benutzerauthentifizierung notwendig ist. Danach können lokale Ressourcen, wie Festplattenkapazität oder Logdateien mit dem entsprechenden Plugin direkt auf dem entfernten Rechner überwacht werden. Damit der Client diese Plugins verwenden kann, müssen sich die gewünschten Plugins auf dem zu überwachendem Computer befinden. Eine beispielhafte Verwendung mit dem dafür gedachten Nagios-Plugin „*check_by_ssh*“ wird in Abbildung 19 gezeigt.

²²Quelle: [Barth08] S. 98

```
paul@iurpaul:/usr/lib/nagios/plugins$ ./check_by_ssh -Hexample.kit.edu-C "/bin/check_swap -w 20 -c 10"
SWAP OK - 100% free (384 MB out of 384 MB) |swap=384MB;0;0;384
```

Abbildung 19: Manuelle Ausführung eines Servicechecks über SSH

Methode 3 - NRPE Eine alternative Möglichkeit solche Dienste auf entfernten Rechnern zu überwachen, ist durch den sogenannten Nagios Remote Plugin Executor (**NRPE**). Hier muss auf dem Client ein „Agent“ installiert werden, welcher einen Port öffnet mit dem der Nagios-Server kommunizieren kann.

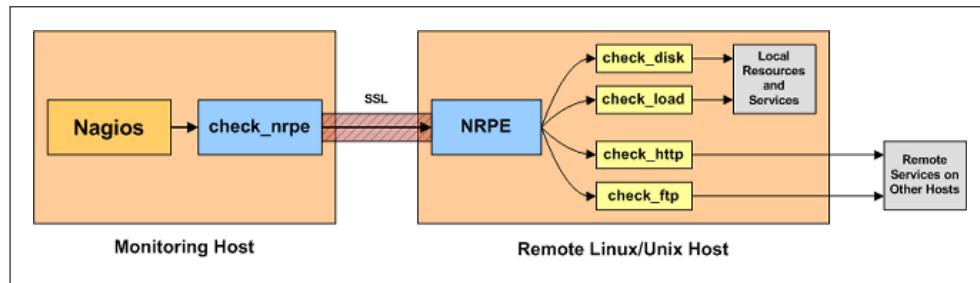


Abbildung 20: Aktive Checks mit NRPE²³

Der Nagios Server kann dann Anforderungen über das Nagios-Plugin „*check_nrpe*“ an den Client verschicken. Ein Aufruf dieses Plugins ist dem des „*check_by_ssh*“ Plugins, siehe Abbildung 19, sehr ähnlich.

Der Nachteil dieser Variante ist ein zusätzlich geöffneter Port und der höhere Aufwand beim Installieren des Agenten im Gegensatz zur Lösung per **SSH**. Zusätzlich gibt es nur die Möglichkeit die Anfragen auf diesem Port auf bestimmte IPs zu beschränken, jedoch nicht den Zugriff durch ein Passwort zu sichern. Dafür beschränkt sich **NRPE** auf die auf dem entfernten Client liegenden Nagios-Plugins und kann nicht System- bzw. Benutzerkommandos aufrufen, wie bspw. das „*rm*“-Kommando zum Löschen von Dateien, welche durch den Einsatz von „*check_by_ssh*“ standardmäßig möglich wären. Sicherheitstechnisch gesehen ist daher die **SSH**-Variante kritischer, da es einem Angreifer ermöglicht auf System- bzw. Benutzerkommandos zuzugreifen, wenn

²³Quelle: [Nagios]

er die Kontrolle über den Nagios-Server erlangt. Beide Verfahren unterstützen die Verschlüsselung der Datenübertragung zwischen Nagios-Server und Client, so dass keine Informationen im Klartext übertragen werden.

Methode 4 - SNMP Diese Variante wird nur verkürzt behandelt, da sich diese Arbeit hauptsächlich mit der Überwachung von Servern beschäftigt und nicht von Netzwerkkomponenten wie Switches oder Router, die nur durch das Simple Network Management Protocol (**SNMP**) überwacht werden können, wenn mehr Informationen als eine schlichte Erreichbarkeit gesammelt werden sollen.

Barth schreibt über diese Variante:

„Mit dem Simple Network Management Protocol **SNMP** lassen sich ebenfalls lokale Ressourcen übers Netz abfragen [...]. Ist auf dem Zielhost ein **SNMP**-Daemon installiert [...] kann Nagios ihn nutzen, um lokale Ressourcen wie Prozesse, Festplatten oder Interface-Auslastung abzufragen.“

[Barth08] S. 101

Durch **SNMP** kann auf die strukturierte Datenhaltung der **MIB**²⁴ in den entfernten Netzwerkknoten zugegriffen werden. Der Aufbau einer **MIB** wird in Abbildung 21 gezeigt. Anhand dieser Anordnung können die **SNMP**-Plugins von Nagios den gewünschten Wert über das Netzwerk abfragen.

Bei einem Switch werden die auslesbaren Informationen vom Hersteller bestimmt. Wenn auf einem Rechner eigene Ergebnisse in der **MIB** abgespeichert werden sollen, muss dies durch einen **SNMP**-Daemon eingetragen werden.

²⁴Die Management Information Base (**MIB**) dient als **SNMP**-Informationstruktur und besteht aus einem hierarchischen, aus Zahlen aufgebauten Namensraum. Ähnliche Struktur wie andere hierarchische Verzeichnisdiensten wie **DNS** oder **LDAP**. Quelle: [Barth08] S.233

²⁵Quelle: [Munin08] S. 156

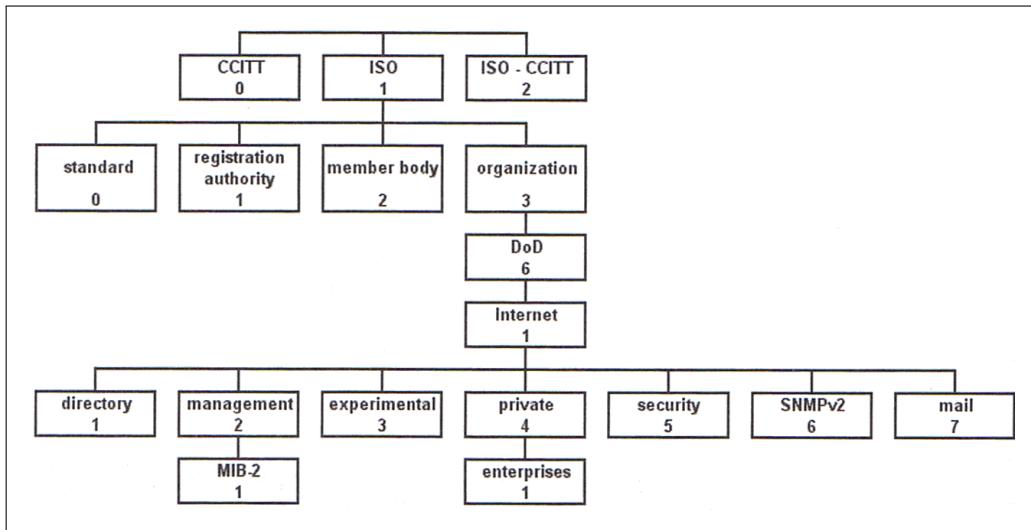


Abbildung 21: Struktur der Management Information Base²⁵

Dessen Konfiguration ist im Vergleich zu den anderen Überwachungsmethoden deutlich komplexer.

Es gibt zwei verschiedene Möglichkeiten Dienste mit **SNMP** zu überwachen. Der Nagios-Server fragt aktiv den Inhalt der entsprechenden **MIB**-Einträgen periodisch ab oder der Client sendet asynchron seine Statusmeldungen über **SNMP** an Nagios. Bei der letzteren passiven Variante spricht man auch von sogenannten **SNMP**-Traps.

Methode 5 - NSCA Diese Methode verwendet passive Checks. Bei passiven Tests führt der zu überwachende Computer das statuserzeugende Plugin selbst aus und sendet es über ein weiteres Plugin zum Nagios-Server. Hierfür muss das Testprogramm bzw. Script und das entsprechende Plugin „*send_ncsa*“, welches zum Versenden der Informationen zuständig ist, auf dem Host vorhanden sein. Auf der anderen Seite muss der „**NSCA**“ (Nagios Service Check Acceptor) auf dem Nagios-Server als Daemon gestartet sein, damit die übermittelten Ergebnisse von Nagios entgegengenommen werden.

Folgende Abbildung soll das Prinzip der passiven Checks verdeutlichen:

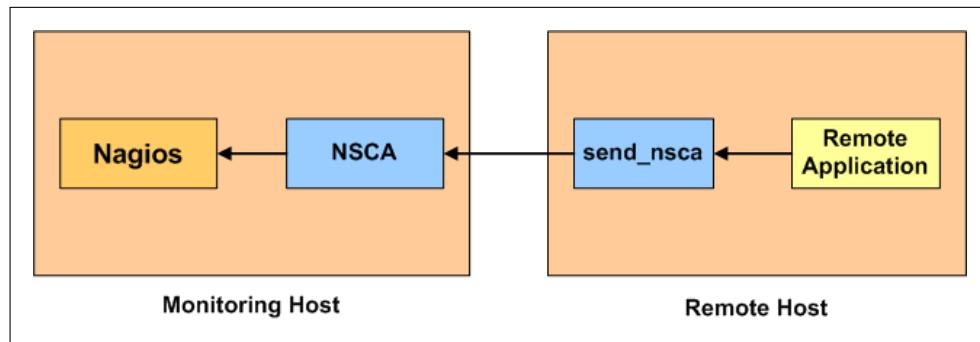


Abbildung 22: Passive Checks mit NSCA²⁶

Das Testprogramm *Remote Application* wird selbstständig vom zu überwachten Rechner *Remote Host* aufgerufen und übermittelt durch das „*send_ncsa*“ Plugin die Ergebnisse über das Netzwerk an den Nagios-Server *Monitoring Host*. Da auf diesem der NSCA als Daemon läuft können die Ergebnisse an die Nagios-Anwendung zur Auswertung weitergegeben werden.

Die Wertung der verschiedenen Methoden und die Auswahl für die Umsetzung wird in Kapitel 7.2 beschrieben.

²⁶Quelle: [Nagios]

5 Oracle UCM

5.1 Allgemein

Oracle Universal Content Management basiert auf der Software Stellent von der gleichnamigen Firma Stellent, welche im November 2006²⁷ von Oracle gekauft wurde.

5.2 Aufbau

Die Architektur des **Oracle UCM**-Systems gliedert sich in separate Komponenten auf wie in Abbildung 23 gezeigt wird.

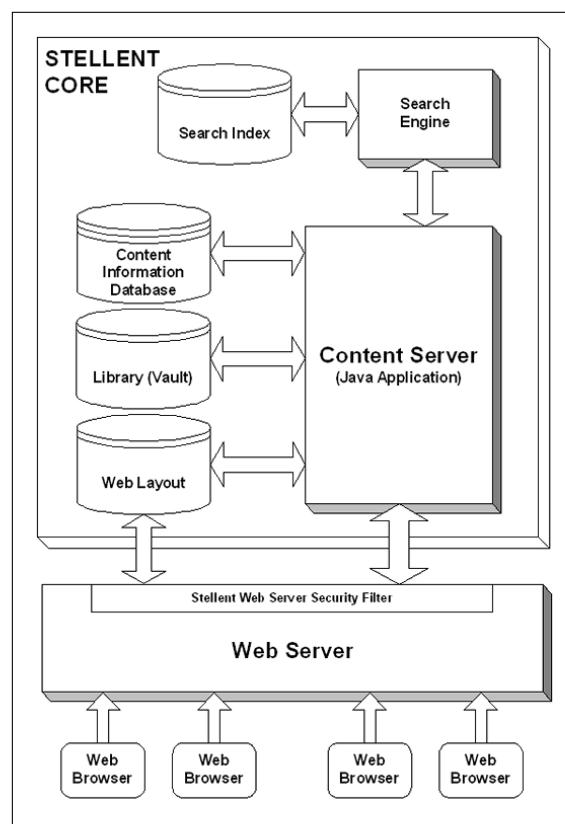


Abbildung 23: Oracle UCM Architektur²⁸

²⁷Quelle: [OraPress]

²⁸Quelle: [ClubOra]

Die Anwendung **Oracle UCM** ist aus folgenden Kernkomponenten aufgebaut:

Content Server Der Content Server ist das Herzstück der Oracle UCM Anwendung und basiert auf einer Java-Anwendung. Er dient als Grundgerüst (Framework) für darüber liegende Funktionen, da er für die Ablage der Dokumente sowie deren Verwaltung, siehe Abbildung 5, verantwortlich ist.

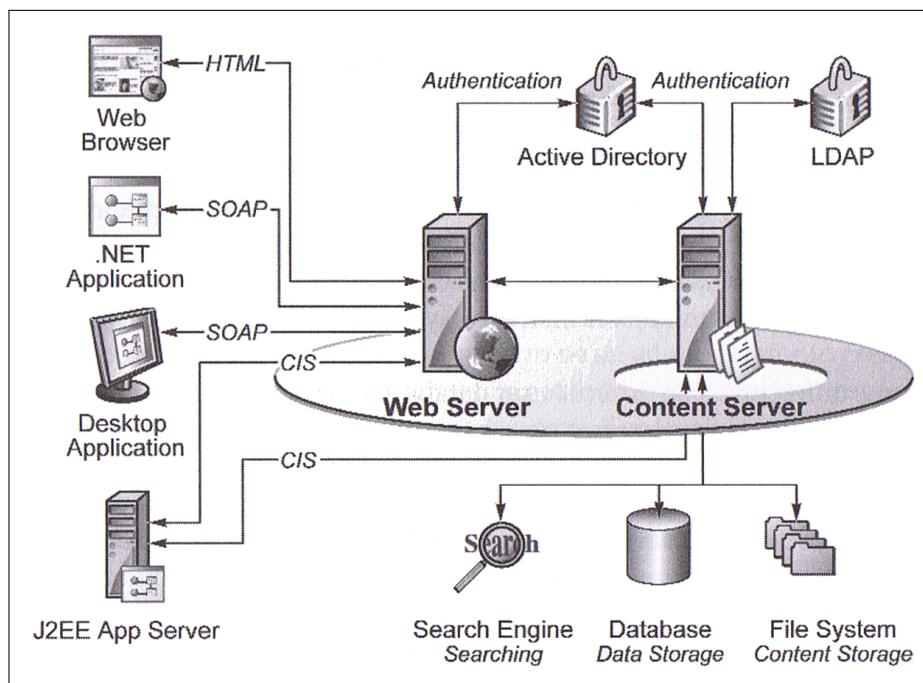


Abbildung 24: Beispielhafter Einsatz eines Content Servers²⁹

Dieses Framework ist als Service-Oriented Architecture (**SOA**) aufgebaut. Im Kontext des Content Servers wird als Service ein diskreter Aufruf einer Funktion verstanden. Dabei kann diese Funktion das Hinzufügen, die Bearbeitung, die Konvertierung oder das Herunterladen eines Dokumentes bedeuten. Diese Services und ihre einzelnen Funktionen werden durch das **SOA**-Framework verdeckt und stehen als Web Service zur Verfügung.

Inbound Refinery Die Inbound Refinery ist für die Konvertierung der Dokumente zuständig und ist keine interne Komponente des Content Ser-

²⁹Quelle: [Huff06] S. 17

vers, sondern kann sich auch auf einem anderen Server befinden. Dabei werden spezielle Add-ons (Filter) für die Konvertierung verwendet. In zeitlichen Abständen überprüft die Inbound Refinery, ob die bisher eingechckten Dokumente konvertiert werden müssen, und speichert die konvertierte Datei in den Web Layout-Ordner.

Data Storage Der Content Server verwaltet die Datenbank, die die Metadaten über die Dokumente beinhaltet. Diese Metadaten werden für die Versionierung, Verwaltung und Suchanfragen verwendet.

Content Storage Der Content Storage liegt auf dem Dateisystem und ist in Vault und Web Layout aufgeteilt.

Vault und Web Layout Der **Vault** ist ein Ordner auf dem Server in dem die Originaldateien der Benutzer in ihrem nativen Format gespeichert werden. Im Gegensatz dazu werden im **Web Layout** die konvertierten Versionen der Dokumente abgelegt. Beispielsweise eine **PDF**-Version einer Microsoft Word-Datei.

Search Engine Eine Suchanfrage eines Benutzers wird zuerst an den Webserver gesendet, der die Anfrage an den Content Server weitergibt. Der Content Server verwendet anschließend seine Search Engine um ein Suchergebnis zu erhalten. Das Suchergebnis wird dem Webserver übergeben, der das Ergebnis an den Benutzer sendet. Die Search Engine verwendet einen Suchindex, der aus den Metadaten und Referenzen zu den Volltextversionen der Dokumente besteht.

Webserver Der Webserver ist hauptsächlich für die Präsentation und Ausgabe der gespeicherten Dokumenten und Informationen zuständig. Dabei ist er auch für die Authentifizierung der Benutzer zuständig.

5.3 Konkrete Verwendung

Oracle UCM wird als Enterprise-Content-Management für die Verwaltung von Webseiten, Dokumenten und Bildern im Forschungszentrum Karlsruhe eingesetzt.

Dabei wird im konkreten Anwendungsfall Oracle UCM als Bilddatenbank verwendet. Diese Bilddatenbank nimmt Fotos und Bilder der Benutzer entgegen (*Einchecken*) und konvertiert das Originalbild dabei in andere Bildversionen wie beispielsweise eine verkleinerte Version für Webseiten.

Dieser typische Ablauf soll durch Abbildung 25 verdeutlicht werden.

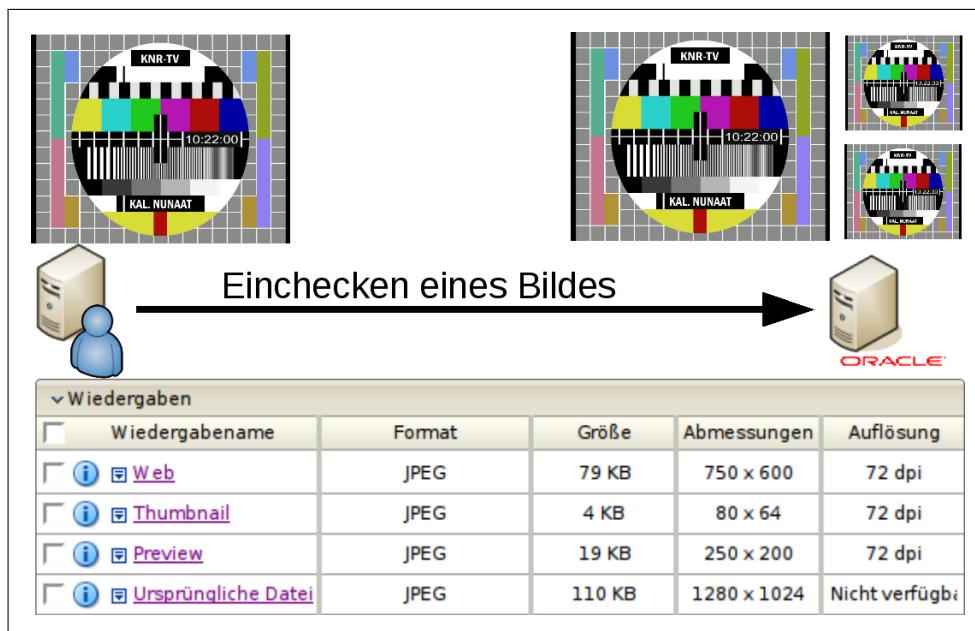


Abbildung 25: Bilddatenbank als Anwendung

Die untere Tabelle zeigt die verschiedenen Bildversionen einer bereits konvertierten Bilddatei.

Da die Bilddatenbank unter einem Windows-Server betrieben werden soll, muss dies für die Überwachung bei der Auswahl der Überwachungselemente und Realisierung der Überwachung durch Nagios berücksichtigt werden.

6 Überwachungselemente

Die Überwachung eines Dienstes über ein Netzwerk verteilt sich auf verschiedenen Ebenen mit unterschiedlichen Gewichtungen. Zum Beispiel stellt das simple Senden eines Pings an den entsprechenden Server die niedrigste und primitivste Stufe dar, da hier lediglich die Netzwerkschnittstelle des Servers auf ihre Funktionalität und dabei der Status der Netzwerkstrecke getestet wird. Ob die Anwendung überhaupt auf dem Server läuft und in welchem Zustand sie sich befindet, muss auf eine andere Weise herausgefunden werden.

Dabei lassen sich aus den verschiedenen Überwachungselementen vier Kategorien Statusabfragen, Funktionalitätstests, Auswertung von Logdateien und Benutzersimulation bilden.

6.1 Statusabfragen

Diese Kategorie besteht aus einfacheren Überprüfungen, die jeweils den Status des Überwachungselementes überwachen. Dabei können weitere Untergruppen gebildet werden:

System

- **Ping** Überprüft, ob der Rechner vom Nagios-Server über das Netzwerk erreichbar ist.
- **Prozessorauslastung** Überwacht die Auslastung des Prozessors und schlägt bei ungewöhnlich hohen Werten Alarm.
- **Festplattenspeicherausnutzung** Überwacht die Speicherplatzauslastungen der verschiedenen Festplattenpartitionen, damit immer genügend Speicherplatz für Anwendungen und Betriebssystem verfügbar ist.

- **Arbeitsspeicherauslastung** Beobachtet wie viel Arbeitsspeicher vom System verwendet wird und wie viel davon noch zur Verfügung steht.

Prozesse

- **IdcServerNT.exe** Der Prozess der **Oracle UCM**-Awendung.
- **IdcAdminNT.exe** Der Prozess für das Administration-Webinterface von **Oracle UCM**.
- **w3wp.exe** Der Prozess des Webservers Microsoft „Internet Information Services“

Services

- **IdcContentService** Den Zustand des Content Server-Dienstes „*scdms01*“ überprüfen.
- **IdcAdminService** Den Dienst für die Content Server-Administration „*sccdms01_admin*“ überwachen.
- **Zeitsynchronisationsdienst** Überprüfen, ob der „*W32TIME*“-Dienst, der für den Zeitabgleich mit einem Zeitserver zuständig ist, läuft und die Abweichung zwischen Client und Zeitserver festhalten.
- **Antivirusdienst** Den Zustand des Dienstes „*Symantec AntiVirus*“ überprüfen, der für die Updates des Virusscanners notwendig ist.

6.2 Überwachung der Funktionalität

Durch die vorherigen Tests kann herausgefunden werden, ob eine Anwendung oder ein Dienst auf dem Server gestartet wurde. Die Funktionalität kann durch solche Überprüfungen jedoch **nicht** sichergestellt werden. Da beispielsweise der Prozess bzw. Dienst des Webservers gestartet ist, jedoch

keine Webseite aufgerufen werden kann. Daher muss eine weitere Art von Überprüfungen die Anwendungen auf ihre Funktionalität testen.

- **Webserver** Aufruf einer Webseite auf dem Server. Wenn auf diese Anfrage eine gültige Antwort in Form einer Statuscode-Meldung erfolgt, kann die korrekte Funktion des Webservers festgestellt werden.
- **Webinterface des Oracle UCM** Zusätzlich wird mit dieser Abfrage die Integration des Content-Management-Systems in den Webserver überwacht, da hier nicht nur der Webserver, sondern eine **Oracle UCM** spezifische Webseite abgefragt wird.
- **Benutzeranmeldung am Oracle UCM** Hier wird getestet, ob sich ein Benutzer erfolgreich am System anmelden kann. Dies wird mit Anmeldungsdaten eines lokalen Benutzers und eines Active Directory-Benutzers durchgeführt um gleichzeitig die Verbindung zum Active Directory-Server zu testen.
- **Oracle Datenbank** Überprüft den Verbindungsauflauf zur Datenbank. Wenn keine Verbindung zur Oracle-Datenbank möglich ist, können keine neuen Informationen gespeichert werden.
- **Anzahl Datenbankverbindungen** Anzahl der Verbindungen zur Datenbank, da aus Performanzgründen eine Obergrenze mit einer maximalen Anzahl festgelegt ist.

6.3 Auswerten von Logdateien

In dieser Kategorie werden zusätzlich verschiedene Logdateien auf spezielle Warnungs- und Fehlermeldungen anhand eindeutigen Stopwörtern untersucht. Dies ist notwendig um reaktiv Fehlverhalten der Anwendung zu erkennen, das nicht mit den vorherigen Überwachungselementen entdeckt wurde. Des weiteren können durch die Analyse der Logdateien etwaige Alarm-

meldungen der bisherigen Tests bestätigt, begründet oder aufgehoben werden. Somit bietet das Auswerten der Logdateien zusätzliche Sicherheit False Positive- oder False Negative-Meldungen auszuschließen.

Die Oracle UCM Anwendung erstellt drei verschiedene Arten von Logdateien.³⁰

- Content Server Log
- Inbound Refinery Log
- Archiver Log

Um alle Logs ohne Probleme im Internetbrowser anzuzeigen, liegen alle Logdateien im HTML-Format vor. Alle drei Arten von Logs bestehen jeweils aus 30 verschiedenen Dateien, die sich täglich abwechseln. Dadurch wird für jeden Tag im Monat eine separate Datei verwendet, um bei vielen Warnungs- und Fehlermeldungen durch die chronologische Anordnung den Überblick zu behalten. Dabei werden die Logdateien zwangsweise nach 30 Tagen nacheinander überschrieben.

Diese Rotation der Logdateien muss bei der Durchsuchung nach Stopwörter beachtet werden, damit stets die aktuelle Logdatei überwacht wird und keine veralteten Informationen für False Positive-Meldungen durch Nagios sorgen.

6.4 Benutzersimulation

Ein Dokument nimmt in seinem Lebenszyklus, siehe Abbildung 5, verschiedene Zustände an. So kommt es nach dem Einchecken in den Zustand „*genwww*“. Der darauf folgende Zustand „*fertig*“ gibt die erfolgreiche Konvertierung des Dokumentes bekannt. Die folgende Indizierung wird durch den Zustand „*frei-gegeben*“ angezeigt.

Die Benutzersimulation hat die Aufgabe alle Schritte der Zustandsänderung durch typische Abfragen zu überprüfen.

³⁰Quelle: [UCMlog09]

- **Einchecken von Dokumenten** Damit die eigentliche Aufgabe des Dokumentenverwaltungssystem überwacht werden kann, werden verschiedene Datenformate testweise eingezcheckt. Dabei wird die Antwort der Anwendung auf das Hinzufügen der Dateien analysiert.
- **Konvertierung** Da das hinzugefügte Dokument nicht nur einfach auf dem Server gespeichert wird, sondern dabei auch in ein anderes Format umgewandelt wird, muss diese Konvertierung zusätzlich überwacht werden. Wird beispielsweise ein Bild eingezcheckt, wird dieses mehrfach in verschiedenen Auflösungen oder in einem anderen Bilddateiformat gespeichert. Ob diese Transformation erfolgreich ablief, kann anhand dieser neuen Dateien festgestellt werden.
- **Indizierung** Bei dem Einchecken sollen auch gleichzeitig zusätzliche Informationen über das Dokument festgehalten werden. Diese Informationen können beispielsweise der Name des Autors, das Erstellungsdatum der Datei oder - bei Bildern - der verwendete Farbraum sein. Bei der Suche nach einem Dokument können diese Informationen als zusätzliche Suchkriterien verwendet werden. Daher muss überprüft werden, ob diese Dateien richtig ausgelesen, der Datenbank hinzugefügt und vom Anwender abgefragt werden können. Dabei werden auch zuvor ausgewählte Testdateien verwendet.
- **Suchfunktion** Nach einer erfolgreichen Indizierung muss das eingezcheckte Dokument per Suchanfrage gefunden werden. Dabei wird der Suchbegriff an dem Dateinamen des Testbildes festgelegt.

6.5 Zusammenfassung

Die Basis für die alle anderen Tests bildet die Systemüberwachung. An erster Stelle der Systemüberwachung steht die schlichte Erreichbarkeit über das Netzwerk per Ping. Wenn der Server nicht erreichbar ist, können auch kei-

ne weiterführende Prüfungen durchgeführt werden. Zur Systemüberwachung gehören auch allgemeine Informationen über die Systemressourcen wie freier Festplattenspeicher oder Prozessorauslastung. Die nächste Stufe bildet die Überprüfung der laufenden Prozesse und der Status verschiedener Dienste bzw. Services. Sollten bestimmte Prozesse nicht gefunden werden oder wichtige Dienste nicht gestartet sein, können auf diese Prozesse und Dienste aufbauende Checks nicht funktionieren. Beispielweise kann der Funktionalitätstest der Benutzeranmeldung nicht realisierbar sein, wenn bereits zuvor in der Systemüberwachung der Prozess für den Webserver **IIS** nicht gefunden werden konnte.

Alle Überwachungselemente lassen sich inklusive ihrer Abhängigkeiten in Form der Pyramide (Abbildung 26) darstellen.

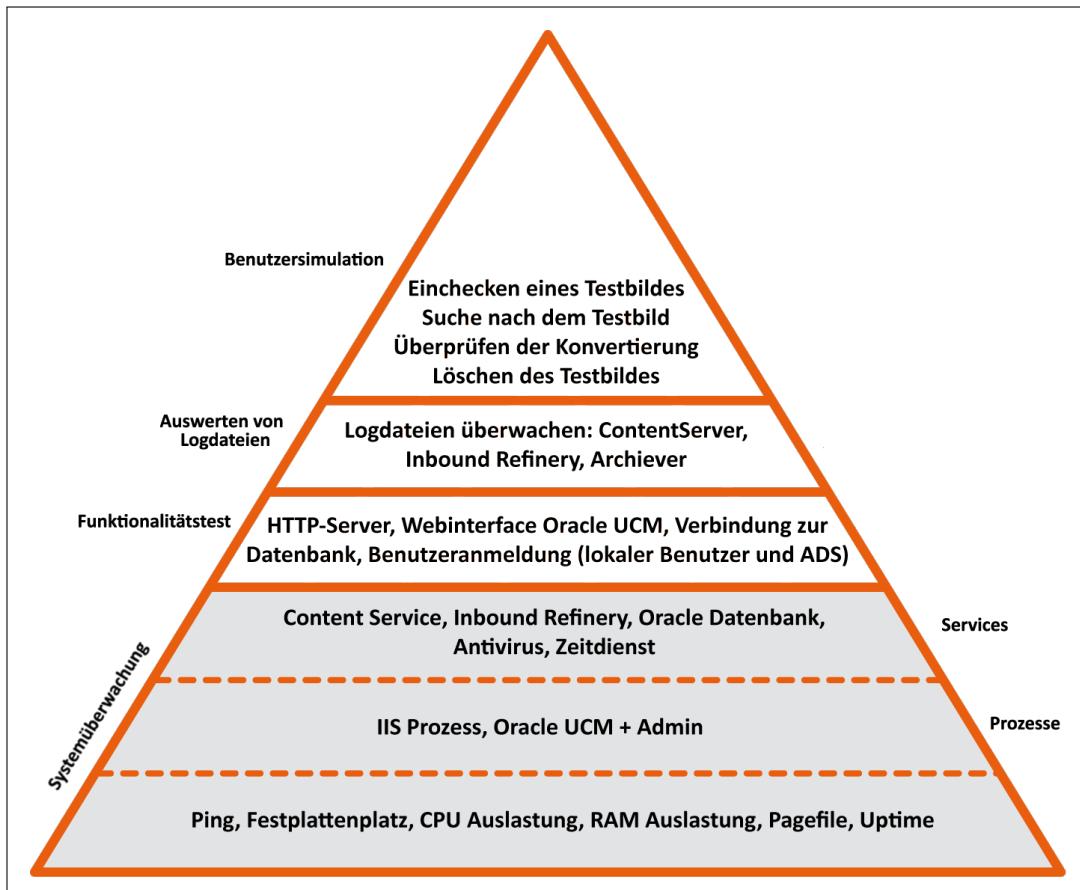


Abbildung 26: Überwachungselemente

7 Umsetzung

In diesem Kapitel wird die Vorgehensweise der zuvor beschriebenen Problemstellungen erörtert.

7.1 Aufbau der Testumgebung

Die für die Umsetzung notwendigen Ressourcen in Form eines Test-Servers und einer virtuellen Maschine.

Aufsetzen eines Nagios-Test-Systems Da die einzelnen Überwachungselemente in der Überwachungssoftware Nagios sukzessiv eingetragen werden müssen, ist ein häufiges Neustarten der Nagios-Anwendung notwendig, damit die neuen Konfigurationsdateien übernommen werden.

Damit dies nicht auf dem bereits verwendetem Nagios-Server durchgeführt werden muss, wird ein Nagios-Testserver für diesen Zweck eingesetzt.

Da Nagios ein Unix-ähnliches Betriebssystem erfordert, wird für diesen Zweck die Linux-Distribution Debian als Betriebssystem des Testservers verwendet. Diese Distribution wird auch auf den Produktivservern des KIT verwendet.

Bilddatenbank als virtuelle Maschine Für die Simulation der verschiedenen Fehlerzuständen der einzelnen Überwachungselemente wird eine virtuelle Maschine mit einer **Oracle UCM** Prototypinstalltion als Entwicklungsplattform verwendet.

7.2 Auswahl der geeigneten Überwachungsmethode

Wie in Kapitel 4.3 aufgeführt, bietet Nagios verschiedene Möglichkeiten Informationen über zu überwachende Objekte zu sammeln.

- Überwachung direkt über das Netzwerk
- Ausführung der Plugins durch **SSH**-Verbindung

- Ausführung von selbst vorkonfigurierten Kommandos durch **NRPE**
- Abfrage von Informationen durch **SNMP**
- Passiver Erhalt der Ergebnisse durch **NSCA**

Für Dienste, die sich über das Netzwerk erreichen lassen, können die dafür entwickelten Nagios-Plugins direkt vom Nagios-Server verwendet werden. Da Windows als Betriebssystem auf dem **Oracle UCM**-Server eingesetzt wird, kann die **SSH**-Variante nicht eingesetzt werden. Die **NRPE**-Methode wird für die Auswertung der Logdateien verwendet. Die Überwachung per **SNMP** wird im Karlsruhe Institute of Technology nicht eingesetzt, da man eine **DoS**-Attacke durch das Senden von vielen **SNMP**-Traps zu dem Nagios-Server verhindern will. Die passive Vorgehensweise mit **NSCA** wird für die Umsetzung nicht benötigt und deshalb nicht verwendet.

Für die Überwachung von Windows-Servern wurde eine weitere Methode entwickelt, die auf dem Prinzip von **NRPE** basiert. Diese Variante wird NSClient genannt und benötigt, wie **NRPE**, die Installation eines Nagios-Agenten auf dem zu überwachenden Server. Daher muss eine Übersicht über verschiedene Nagios-Agenten erstellt werden.

7.3 Übersicht Nagios-Agenten

In diesem Unterkapitel werden die populärsten Agenten für Unix und Windows Betriebssysteme aufgelistet und nach den Punkten Sicherheit, subjektiver Aufwand für die Konfiguration und Art der Abfragemethode (aktiv oder passiv) verglichen.

7.3.1 Unix-Agenten

Für die auf Unix basierenden Betriebssysteme werden fünf verschiedene Möglichkeiten angeboten, die in Abbildung 17 als verschiedene Überwachungsmöglichkeiten von Nagios aufgelistet wurden.

	<i>SSH</i>	<i>NRPE</i>	<i>SNMP</i>	<i>SNMP Traps</i>	<i>NSCA</i>
Methode					
<i>aktiv</i>	✓	✓	✓	-	-
<i>passiv</i>	-	-	-	✓	✓
Sicherheit					
<i>Passwort</i>	-	-	✓(v3)	✓(v3)	-
<i>Accesslist</i> ¹	✓ ²	✓	✓(v2)	✓(v2)	✓
<i>Verschlüsselung</i>	✓	✓	✓(v3)	✓(v3)	✓
Aufwand ³					
	niedrig	normal	hoch	hoch	normal

¹ Einschränkung der Abfrage der Überwachungsinformationen anhand der IP-Adresse

² Über zuvor ausgetauschte SSH-Schlüssel

³ Subjektive Einschätzung

Tabelle 2: Übersicht der verschiedenen Unix-Agenten

Dabei werden drei Agenten genannt, die eine aktive Ausführung der Nagios-Plugins benutzten. Alle drei Agenten unterscheiden sich jedoch in den Punkten Sicherheit und Aufwand. Der auf *SSH* basierende Agent besitzt einen relativ geringen Aufwand für die Installation, da für den Aufbau der Kommunikation zwischen Nagios-Server und Client nur der öffentliche Schlüssel des Servers auf dem Client eingetragen werden muss. Dadurch kann der Nagios-Server sich ohne Passwortabfrage an dem zu überwachendem Host anmelden und die sich darauf befindlichen Nagios-Plugins ausführen. Da auf den meisten Unix-Servern bereits ein *SSH*-Server läuft und deshalb kein weiterer Port geöffnet oder eine weitere Software installiert werden muss, ist diese Methode den anderen meist vorzuziehen.

Bei der *NRPE*-Methode wird eine weitere Softwarekomponente auf dem Client installiert, die einen separaten Port für die Kommunikation mit dem

Nagios-Server öffnet. Wie bei dem Aufruf per **SSH** müssen sich hier die Nagios-Plugins bereits auf dem Rechner befinden. Dabei gilt als Unterschied dieser zwei ähnlichen Methoden zu beachten, dass für die Ausführung der Checks per **SSH** ein extra Benutzerkonto auf dem Client erstellt werden muss und somit ohne Anpassung der Benutzerrechte Systembefehle ausgeführt werden können, während die Ausführung von Kommandos bei **NRPE** nur auf vorkonfigurierte Befehle beschränkt ist, wie in Kapitel 4.3.2 aufgeführt. Da **SNMP** plattformunabhängig funktioniert ist es möglich diese Variante bei Unix- sowie bei Windowsservern einzusetzen. Die verwendete **SNMP**-Version bestimmt welche Sicherheitsmerkmale zur Verfügung stehen. Zwar gibt es bereits seit Version 1 die Möglichkeit den Zugriff per Passwort in drei Gruppen aufzuteilen: kein Zugriff, Leserecht und Lese- mit Schreibrecht³¹, jedoch wird dieses Passwort im Klartext übertragen, so dass es leicht auslesbar ist. Auch die **SNMP**-Version 2 inklusive der erweiterten Version 2c verwendet die gleiche unsichere Authentifizierung. Erst ab Version 3 wird das Passwort verschlüsselt übertragen. Während Barth behauptet, dass man bei **SNMP** generell kein Passwort verwenden soll³², da es leicht per Netzwerkmitschnittprogramme, wie Wireshark, ausgelesen werden kann, wird in [Jose07] S. 121 klargestellt, dass die Version 3 eine verschlüsselte Authentifizierung durch den MD5- oder SHA-Algorithmus ermöglicht.

Die passive Variante über **SNMP** bei der der Client die Ergebnisse der Checks an den Nagios-Server sendet, auch **SNMP**-Traps genannt, funktioniert nach dem gleichen Prinzip. Da das Auslesen der **MIB** per **SNMP** im Gegensatz zu den anderen Varianten deutlich komplexer ist, wird der Aufwand als hoch eingestuft, siehe Bewertung in Kapitel 7.2.

Ein weiterer Vertreter, der passive Checks ermöglicht, ist der **NSCA**-Agent. Wie die anderen Unix-Agenten bietet es die Möglichkeit den Datenaustausch

³¹Quelle: [Barth08] S. 237

³²Quelle: [Barth08] S. 238

zwischen Nagios-Server und Client zu verschlüsseln. Alle Unix-Agenten erlauben es den Zugriff auf die Nagios-Plugins auf bestimmte IP-Adressen zu beschränken. Die Liste mit diesen IP-Adressen nennt man auch *Accesslist*.

7.3.2 Windows-Agenten

Da die zu überwachende Oracle UCM Anwendung auf einem Windows-Server betrieben wird und die bereits vorgestellten Agenten mit Ausnahme der **SNMP**-Varianten nur unter Unix einsetzbar sind, müssen zusätzlich die explizit für Windows entwickelten Nagios-Agenten untersucht werden. Dabei wird die Auswahl der Kandidaten auf vier Bewerber beschränkt, siehe Tabelle 3.

	NSClient	NRPE ^{NT}	NC.net	NSClient++
Methode				
<i>aktiv</i>	✓	✓	✓	✓
<i>passiv</i>	-	-	✓	✓
<i>NSClient</i> ¹	✓	-	✓	✓
<i>NRPE</i> ²	-	✓	✓	✓
Sicherheit				
<i>Passwort</i>	✓	✓	-	✓
<i>Accesslist</i> ³	-	-	✓	✓
<i>Verschlüsselung</i>	-	✓	✓	✓
Aufwand ⁴	normal	hoch	normal	normal

¹ Kompatibilität mit dem NSClient-Dienst

² Erlaubt Ausführung von vorkonfigurierten Kommandos

³ Einschränkung der Abfrage der Überwachungsinformationen anhand der IP-Adresse

⁴ Subjektive Einschätzung

Tabelle 3: Übersicht der verschiedenen Windows-Agenten

Der NSClient-Dienst liefert die Möglichkeit lokale Windows-Ressourcen über das Netzwerk mit eigenem Port (Standort 1248) abzufragen. Das Plugin *check_nt* wurde explizit für diesen NSClient-Dienst entwickelt und steht durch die Nagios-Plugins standardmäßig zur Verfügung. Dadurch können die grundlegende Informationen für die Systemüberwachung aus Kapitel 6.1, wie Zustände von Prozesse, Services, CPU-Auslastung, Festplattenplatz, usw. abgefragt werden. Der Zugriff auf den NSClient-Dienst per *check_nt* wird in Abbildung 27 gezeigt.

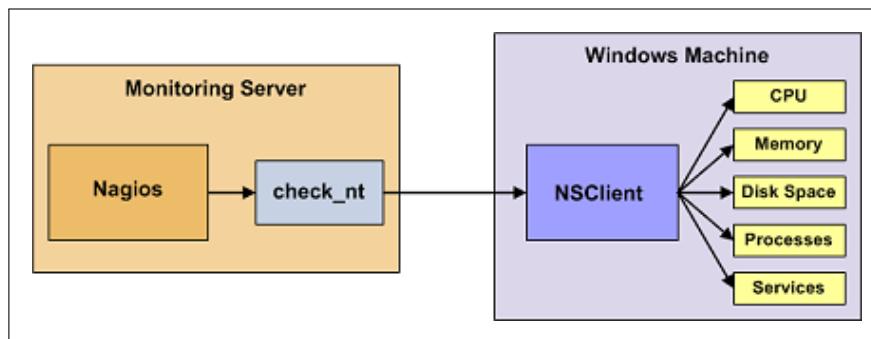


Abbildung 27: Abfrage von Windows-Ressourcen durch *check_nt*³³

Diese Abfrage kann durch die Ausführung auf der Kommandozeile getestet werden:

```
root@nagiosdev:/usr/lib/nagios/plugins# ./check_nt -H example.kit.edu -v CLIENTVERSION
NSClient++ 0.3.6.818 2009-06-14
```

Abbildung 28: Zugriff auf den NSClient-Dienst durch *check_nt*

Der erste und zugleich älteste Agent NSClient wird nicht mehr aktiv entwickelt und ist als aktuellste Version 2.0.1 aus dem Jahre 2003 bereits recht alt. Daher wird auch keine Verschlüsselung der ein- und ausgehenden Daten unterstützt. Auch bietet NSClient keine Möglichkeit aktiv vom Nagios-Server aus Nagios-Plugins oder weite Programme auszuführen, die sich auf dem zu überwachendem Host befinden.

³³Quelle: http://nagios.sourceforge.net/docs/3_0/images/monitoring-windows.png (modifiziert)

Um dies auch für Windows-Server zu ermöglichen gibt es eine auf Windows portierte **NRPE**-Variante, die sich NRPE_NT nennt. Hier lassen sich die Plugins direkt über den Nagios-Server aufrufen und die ausgetauschten Informationen werden verschlüsselt über das Netzwerk übertragen.

Beide bisher genannte Windows-Agenten bieten keine Möglichkeit eine *Accesslist* anzulegen, erst das Programm NC_net bietet diese Möglichkeit inklusive dem Sicherheitsmerkmal Verschlüsselung an. Außerdem können durch den eingebauten **NRPE**-Dienst aktiv Nagios-Plugins auf dem Client aufgerufen werden. Als Besonderheit lassen sich durch NC_net sowohl aktiv als auch passiv Testergebnisse an den Nagios-Server übertragen.

Der Nagios-Agent NSClient++ besitzt diesselben Merkmale wie NC_net, jedoch kann der Nagios-Server noch über ein Passwort zusätzlich verifiziert werden.

Die Möglichkeit Informationen per **SNMP** und **SNMP**-Traps abzufragen ist auch unter Windows möglich. Dabei gelten die gleichen Richtlinien, Hinweise und Einschränkungen wie zuvor in Kapitel [7.3.1](#) aufgeführt.

7.3.3 Auswahl und Konfiguration des Nagios-Agenten

Auswahl Anhand der im vorherigen Kapitel beschriebenen Übersicht der Windows-Agenten und der daraus resultierenden Übersichtstabelle [3](#) wird ein geeigneter Kandidat für die Testumgebung ausgewählt. Da nur ein Windows-Agent alle drei Sicherheitsmerkmale anbietet und dabei aktive und passive Überwachungsmethoden erlaubt, fällt die Wahl **NSClient++**.

Installation und Konfiguration Der Nagios-Agent NSClient++ kann im Gegensatz zu den meisten anderen Windows-Agenten komfortabel über einen graphischen Benutzerdialog installiert werden. Während des Installationsvorgangs kann auch festgelegt werden welche Komponenten installiert werden sollen. Dabei werden diese Komponenten nicht standardmäßig geladen.

den, sondern im nächsten Dialogfenster auswählbar:

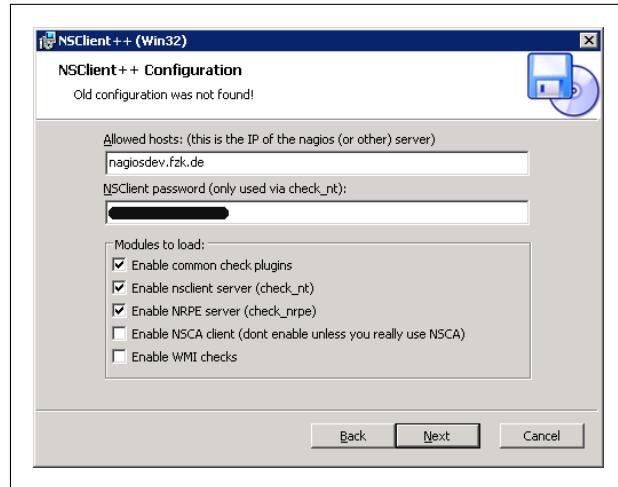


Abbildung 29: Konfiguration des NSClient++ während der Installation

Außerdem können direkt während der Installation die **IP**-Adresse bzw. der **FQDN** des Nagios-Servers und das gewünschte Passwort eingetragen werden. Durch die während des Installationsprozesses geladenen Komponenten für den NSClient- und **NRPE**-Dienst können die Standard-Nagios-Plugins *check_nt* und *check_nrpe* mit dem Windows-Server verwendet werden. Dabei läuft die Kommunikation zwischen dem Nagios- und dem Windows-Server folgendermaßen ab:

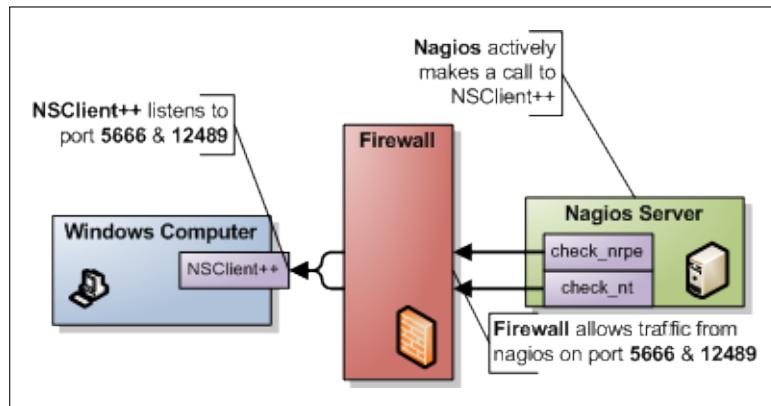


Abbildung 30: Kommunikation zwischen Nagios und NSClient++³⁴

³⁴Quelle: [NSClient]

Bei Windows-Server mit vielen Verbindungen und Diensten können Remote Procedure Calls (**RPC**), bei denen dynamisch Ports ab 1025 verwendet werden, bereits den Standardport des NSClient-Dienstes (1248) unter Umständen bereits vor dem Start des Dienstes belegen.³⁵ Um dies zu verhindern wurde der Port des NSClient-Dienstes beim NSClient++ bereits vom Entwickler auf einen höheren Port (12489) gewechselt.

Alle bisherigen Einstellungen können in der Konfigurationsdatei *NSC.ini*, die sich in dem Installationsverzeichnis des NSClient++ befindet, verändert werden. In dieser Datei befinden sich noch mehr Einstellungsmöglichkeiten; im Folgenden werden nur für die Umsetzung relevanten (notwendig essentiell benötigten) Parameter aufgelistet.

```
1 ;# NSCLIENT PORTNUMMER
2 ;# Die Portnummer des NSClient-Dienstes
3 port=13596
4
5 ;# NRPE PORTNUMMER
6 ;# Die Portnummer des NRPE-Dienstes
7 port=13597
8
9 ;# SSL SOCKET
10 ;# Die Aktivierung von SSL der Kommunikation zwischen Nagios- und Windows-
11 Server
11 use_ssl=1
12
13 ;# NRPE BEFEHLSDEFINITIONEN
14 ;# Definitionen der Befehle, die durch den NRPE-Dienst aufrufbar sind
15 check_uname=scripts\check_uname.exe
16 check_reflog=scripts\check_logfiles.exe -f scripts\logfile.cfg
```

Listing 6: NSClient++ Konfigurationsdatei

Damit die vorgenommen Änderung übernommen werden, muss der Dienst des NSClient++ neu gestartet werden.

Durch das Ausweichen auf höher gelegene Portnummern können die zuvor genannten Probleme aufgrund der **RPCs** verhindert werden.

Der bereits höher liegende Standardport des NSClient-Dienstes beim NSClient++ wird zusätzlich noch abgeändert, damit die Tatsache, dass sich ein Nagios-Agent auf dem Computer befindet, nicht sofort ersichtlich ist. Dieser

³⁵Quelle: [Barth08] S. 481

sicherheitstechnische Aspekt wurde bereits in Kapitel 3.1.3 behandelt.

Damit die SSL-Verschlüsselung zwischen den Servern aktiviert wird, muss man es explizit in der Konfigurationsdatei mit der Option `use_ssl=1` angeben. Die Definitionen der NRPE-Kommandos dienen dafür, dass durch den Nagios-Server per `check_nrpe` mit dem Befehlsnamen der darauf folgende Befehl ausgeführt wird.

Aufgrund der abgeänderten Portnummer muss man den Port bei dem Aufruf explizit angeben. Ein Aufruf eines solchen NRPE-Kommandos vom Nagios-Server wird in der folgenden Abbildung gezeigt:

```
root@nagiosdev:/usr/lib/nagios/plugins# ./check_nrpe -H example.kit.edu -p 13596 -c check_uname
Operating System OK - Microsoft(R) Windows(R) Server 2003 Standard Edition Service Pack 2
```

Abbildung 31: Aufruf eines NRPE-Kommandos

Anhand des Befehlsnamens `check_uname` führt der NRPE-Dienst die in der Konfigurationsdatei eingetragene Datei `check_uname.exe` aus.

Der Aufruf um Informationen durch den NSClient-Dienst abzufragen sieht ähnlich aus:

```
root@nagiosdev:/usr/lib/nagios/plugins# ./check_nt -H example.kit.edu -p 13596 -s secret -v UPTIME
System Uptime - 49 day(s) 21 hour(s) 41 minute(s)
```

Abbildung 32: Aufruf des NSClient-Dienstes

Die Servicedefinition des vorherigen NSClient-Aufrufs muss wie nachfolgend in der Nagios-Konfiguration eingetragen werden:

```
1 define service{
2     use          generic-service
3     host_name   example.kit.edu
4     service_description  Uptime
5     check_command  check_nt!-p 13596 -s secret -v UPTIME
6 }
```

Listing 7: Servicedefinition des NSClient-Checks

Damit nicht jeder einzelne Serviceeintrag abgeändert werden muss, falls sich der Port oder das Passwort des zu überwachenden Computers ändert, können eigene Befehlsdefinitionen erstellt werden.

```

1 define command{
2     command_name      check_nt_example
3     command_line      /usr/lib/nagios/plugins/check_nt -H $HOSTNAME$ -p
4         13597 -p secret -v $ARG1$
```

Listing 8: Server spezifische Befehlsdefinition

Dadurch muss nur diese Befehlsdefinition bei einer Änderung bearbeitet werden. Die vorherige Servicedefinition in Listing 32 kann dann in verkürzter Form eingetragen werden:

```

1 define service{
2     use          generic-service
3     host_name   example.kit.edu
4     service_description Uptime
5     check_command  check_nt_example!UPTIME
6 }
```

Listing 9: Verkürzte Servicedefinition des NSClient-Checks

7.4 Umsetzung der Systemüberwachung

Die in Kapitel 6.1 aufgelisten Prozesse und Services können durch den NSClient-Dienst vom Nagios-Server überwacht werden. Dafür wird der in Listing 8 definierte verkürzte Befehl für *check_nt* benutzt.

```

1 #Prozess des IIS Webservers
2 define service{
3     use          generic-service
4     host_name   example.kit.edu
5     service_description IIS Prozess
6     check_command  check_nt_example!PROCSTATE -1 w3wp.exe
7 }
8
9 #Zeitdienst
10 define service{
11    use          generic-service
12    host_name   example.kit.edu
13    service_description Zeitdienst
14    check_command  check_nt_example!SERVICESTATE -1 W32TIME
15 }
```

Listing 10: Prozess- und Service-Check Servicedefinitionen

Mit diesen zwei Einträgen wird der Prozess des IIS-Webservers und der Status des Dienstes zum Zeitabgleich überwacht. Andere Prozesse und Dienste lassen sich nach dem gleichen Schema überwachen.

Nach einem Neustart von Nagios werden beide Einträge im Webinterface angezeigt:

IIS Prozess	OK	2009-07-22 15:14:08	3d 16h 42m 6s	1/4	w3wp.exe: Running
Zeitdienst	OK	2009-07-22 15:14:42	6d 6h 9m 6s	1/4	W32TIME: Started

Abbildung 33: Prozess- und Dienstüberwachung im Nagios-Webinterface

Die Festplattenspeicherausnutzung und die Prozessorauslastung wird auf ähnliche Weise überwacht. Hierbei muss beachtet werden, dass die Testergebnisse nicht eindeutig sind, im Gegensatz zu der Service- und Prozessüberwachung. Wann Nagios alarmieren soll muss vom Anwender in Form von Parametern festgelegt werden.

```
1 #Belegung der Partition C:
2 define service{
3     use generic-service
4     host_name example.kit.edu
5     service_description C:\ Drive Space
6     check_command check_nt!USEDISKSPACE -l c -w 85 -c
7         100
8 }
9 #CPU Auslastung der letzten 5 Minuten
10 define service{
11     use generic-service
12     host_name example.kit.edu
13     service_description CPU Load
14     check_command check_nt!CPULOAD -l 5,80,100
15 }
```

Listing 11: Überwachung der Festplatten- und Prozessorauslastung

Für diese Festplattenüberwachung versendet Nagios eine Warnung, wenn der belegte Speicherplatz auf der C-Partition die 85% Marke überschreitet und meldet einen kritischen Fehler bei 100%. Bei der Prozessorüberwachung schlägt Nagios Alarm, wenn der Mittelwert der Auslastung in den letzten fünf Minuten mehr als 80% bzw. 100% betragen hat.

7.5 Umsetzung der Funktionitätstest

Für die Ausführung der einfachen Funktionitätstest aus Kapitel 6.2 werden Benutzerinformationen zur Anmeldung benötigt. Nagios besitzt extra hierfür

die Möglichkeit diese Benutzerinformationen in Variablen zu speichern, damit sie nicht einzeln bei jeder Servicedefinition verändert werden müssen. Da sich die Definition dieser Variablen in einer externen Datei befindet, können die Zugriffsrechte auf diese Datei eingeschränkt werden, wodurch die Anmelddaten bei den Servicedefinitionen nicht auslesbar sind.

```

1 #Anmeldung an Oracle UCM mit lokalem Benutzerkonto
2 define service{
3     use generic-service
4     host_name example.kit.edu
5     service_description Anmeldung Oracle UCM als lokaler Benutzer
6     check_command check_http!-u "/bdb/idcplg?IdcService=LOGIN&
7         Action=GetTemplatePage&Page=HOME\_PAGE&Auth=Internet" -a
8         $USER3$:$USER4$ -e "Sie sind angemeldet als" -S
9 }
```

Listing 12: Funktionalitätstest der Benutzeranmeldung

Dabei werden dem Nagios-Plugin *check_http* mit dem „*u*“-Parameter die URL zur Benutzeranmeldungseite und mit dem Parameter „*a*“ der Benutzername und -passwort mitgegeben. Der nach dem Parameter „*e*“ folgende String wird dann in der Antwort des Servers gesucht. Sollte dieser String nicht gefunden werden ist die Authentifizierung fehlgeschlagen und es wird durch Nagios eine Meldung versendet. Mit dem Parameter „*S*“ wird angegeben, dass eine **SSL**-verschlüsselte Verbindung zum Webserver über HTTPS hergestellt werden soll, ansonsten würden die Benutzerinformationen im Klartext übertragen werden, wodurch sie leicht für Angreifer auslesbar wären.

Für das Auslesen von Informationen aus der Statusseite der Oracle UCM-Anwendung wurde ein einfaches BASH-Script entwickelt:

```

1#!/bin/bash
2E_BADARGS=2
3if [ ! -n "$6" ]
4then
5    echo "Usage: `basename $0` -url <URL> -u <username> -p <password>"
6    exit $E_BADARGS
7fi
8
9DBCONNECTIONS=$(wget -qO- --user $4 --password $6 $2 | grep "System
Database")
10DBCONNECTIONS=${DBCONNECTIONS##*>}
11DBCONNECTIONS=${DBCONNECTIONS%% *}
12echo $DBCONNECTIONS
```

Listing 13: Auslesen der Verbindungen zur Datenbank

Dabei muss als URL die Informationsseite mit den Datenbankverbindungen und gültige Benutzerinformationen mitgegeben werden. Anschließend wird die aufgerufene Seite nach der gewünschte Informationen untersucht und ausgetragen.

Dieses einfaches Script kann auch dazu verwendet um andere Informationen von der Statusseite abzufragen.

7.6 Auswertung der Logdateien

Um die drei genannten Logdateien aus Kapitel 6.3 auszuwerten wird durch den **NRPE**-Dienst das Plugin *check_logfiles* von Gerhard Laußer³⁶ eingesetzt. Dieses Plugin besitzt bereits einige nützliche Funktionen für die Überwachung von Logdateien. Durch das Setzen eines Zeitstempels filtert das Plugin veraltete Einträge heraus und untersucht nur neu hinzugekommene Zeilen. Der Rotationsalgorithmus der **Oracle UCM**-Logdateien kann dem Plugin durch die Verwendung einer Konfigurationsdatei mitgeteilt werden.

Die Konfigurationsdatei für das *check_logfiles*-Plugin wird im folgendem Listing gezeigt:

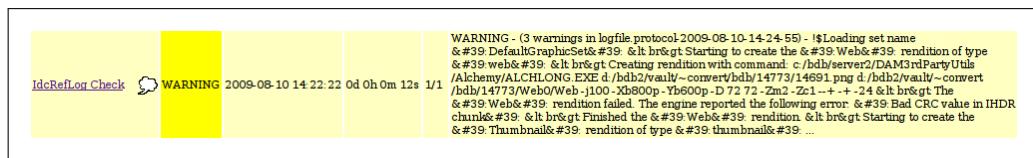
```
1 @searches = ({
2   tag => 'ucmlogs',
3   type => 'rotating::uniform',
4   logfile => 'D:/bdb2/weblayout/groups/secure/logs/bdb/IdcLnLog.htm',
5   rotation => 'refinery\d{2}\.htm',
6   warningpatterns => [
7     'Cannot identify file', #Testbild korrupt
8     'Bad CRC value in IHDR chunk', #Konvertierung nicht erfolgreich
9     'Der Dateiname darf nicht länger als 80 Zeichen sein', #Zu langer
10    Dateiname
11  ],
12});
```

Listing 14: Konfigurationsdatei für *check_logfiles*

Mit dem „tag“-Attribut wird diese Auswertung eindeutig identifizierbar gemacht, da man in der gleichen Konfigurationsdatei mehrere Logdateien bzw. weitere Durchsuchungen definieren kann. Das Attribut „type“ muss hier so gesetzt werden, da die aktuelle Logdatei und die weggrotrierte Logdatei das

³⁶Quelle: <http://www.consol.deopensource/nagios/check-logfiles>

gleichen Namensschema benutzten. Der Pfad zu den Logdateien wird über das Attribut „*logfile*“ gesetzt. Da die einzelnen Logdateien mit einem bestimmten Namensmuster erstellt werden (siehe Kapitel 6.3), muss dieses Namensmuster hier direkt angegeben werden. In diesem Falle durchsucht das *check_logfiles*-Plugin alle Logdateien mit dem Namen *refinery00.htm* bis *refinery99.htm*. Alle gefundenen Dateien werden dem Datum nach sortiert und die aktuellste Datei wird untersucht. Nach welchen Stopwörtern gesucht werden soll wird mit dem Attribut „*warningpatterns*“ angegeben, sofern mindestens einer dieser Strings gefunden wurde liefert das Plugin ein WARNING als Rückmeldung inklusive der Zeile in dem das Stopwort gefunden wurde.



The screenshot shows a terminal window with a yellow header bar containing the text "ldcRefLog Check" and a yellow status bar at the bottom right showing "1/1". The main area displays a single line of text:

```
WARNING - (3 warnings in logfile protocol 2009-08-10 14:24:55) - <!-- Loading set name
&#39;DefaultGraphicsSet&#39; --> Starting to create the &#39;Web&#39; rendition of type
&#39;web&#39;. &lt;br&gt; Creating rendition with command: c:/bdh/server2/DAM3rdPartyUls
/Alchemy/ALCHLONG.EXE d:/bdh2/vault/~convert/obj/b14773/14691.png d:/bdh2/vault/~convert
/bdh/b14773/Web0/Web_j100-Xb800p-Yb600p-D7272-Zc1--+-+--24 &lt;br&gt; The
&#39;Web&#39; rendition failed. The engine reported the following error: &#39;Bad CRC value in IHDR
chunk&#39; &lt;br&gt; Finished the &#39;Web&#39; rendition. &lt;br&gt; Starting to create the
&#39;Thumbnail&#39; rendition of type &#39;thumbnails&#39; ...
```

Abbildung 34: Ausgabe der betreffenden Zeile in der Logdatei

7.7 Benutzersimulation

Um die Funktionalität der Anwendung eindeutig festzustellen werden typische Benutzeraktionen simuliert und die Ergebnisse an Nagios übermittelt. Typische Aktionen sind das Einchecken eines Bildes, Suche nach einem Bild und schließlich die Anforderung des Originalbildes und der konvertierten Bilder.

Da Nagios standardmäßig ein Plugin periodisch jede fünf Minuten aufruft, würde sich die Festplatte und die Datenbank des Oracle UCM-Servers im Laufe der Zeit an ihre Kapazitäten stoßen. Daher werden nach der Anforderung und Überprüfung der Bilder alle Testbilder vom Server entfernt. Ein Testbild soll per Web Service an den Server geschickt und eingechockt werden. Mit der Suche nach dem Testbild und der Anforderung der konvertierten Bilder kann die Funktionalität der Anwendung getestet werden.

Der Ablauf der Benutzersimulation soll in verkürzter Form durch das Struktogramm 35 verdeutlicht werden.

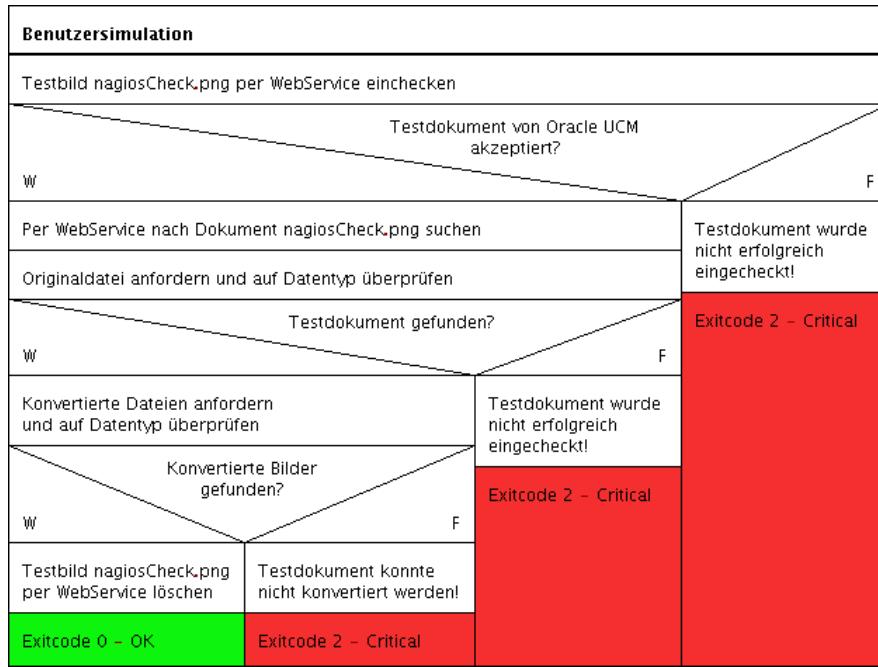


Abbildung 35: Geplanter Ablauf der Benutzersimulation

In diesem ersten Schritt wird durch das Einchecken eines Testbildes die Erreichbarkeit der Anwendung über das Netzwerk getestet.

Wenn die Übertragung des Bildes erfolgreich war, wird anschließend nach dem soeben eingecheckten Bild per Dateinamen gesucht um die Funktionalität der Indizierung zu kontrollieren.

Sollte das Bild gefunden werden, wird es vom Nagios-Server angefordert und auf seine Korrektheit überprüft. Der gleiche Test wird mit den konvertierten Bildversionen durchgeführt, um die Funktion der Konvertierung zu überwachen.

Falls alle Tests erfolgreich waren, wird das Testbild und alle konvertierten Bilder vom **Oracle UCM**-Server gelöscht. Bei den anderen Szenarien gibt das Plugin den Wert 2 für den Status „*CRITICAL*“ zurück.

Die Benutzersimulation wird durch zwei Plugins realisiert.

Einchecken eines Testbildes Das erste Plugin dient zum Einchecken des Testbildes. Dabei ruft der Nagios-Server ein auf **PHP**-basierendes Script auf. In diesem Script wird die **PHP**-Bibliothek nuSOAP eingebunden, damit man vereinfacht auf Web Services zugreifen kann. Die Kommunikation zwischen Client und Server bei der Benutzung eines Web Services findet, wie im Kapitel 3.6 beschrieben, im **XML**-Format statt. Um den Aufwand zu vermeiden diese **XML**-Datei immer selbst zu erstellen, wird mit Hilfe der **WSDL**-Datei auf dem **Oracle UCM**-Server die benötigten Parameter beim Aufruf eines Web Services von nuSOAP ausgelesen.

In der folgenden Abbildung werden aus der **WSDL**-Datei alle möglichen Anforderungsparameter für den Web Service *CheckInUniversal* gezeigt.

```
<s:element name="CheckInUniversal">
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="dDocName" type="s:string"/>
<s:element minOccurs="0" maxOccurs="1" name="dDocTitle" type="s:string"/>
<s:element minOccurs="0" maxOccurs="1" name="dDocType" type="s:string"/>
<s:element minOccurs="0" maxOccurs="1" name="dDocAuthor" type="s:string"/>
<s:element minOccurs="0" maxOccurs="1" name="dSecurityGroup" type="s:string"/>
<s:element minOccurs="0" maxOccurs="1" name="dDocAccount" type="s:string"/>
<s:element minOccurs="0" maxOccurs="1" name="CustomDocMetaData" type="s0:idcPropertyList"/>
<s:element minOccurs="0" maxOccurs="1" name="primaryFile" type="s0:idcFile"/>
<s:element minOccurs="0" maxOccurs="1" name="alternateFile" type="s0:idcFile"/>
<s:element minOccurs="0" maxOccurs="1" name="extraProps" type="s0:idcPropertyList"/>
</s:sequence>
</s:complexType>
</s:element>
```

Abbildung 36: Anforderungsparameter für CheckInUniversal aus der WSDL-Datei

Im PHP-Script werden nach dem Einlesen dieser **WSDL**-Datei und der Authentifizierung am **Oracle UCM**-Server die benötigten Parameter beim Aufruf des Web Services gesetzt und die Ausgabe des Servers ausgewertet.

```
1 $soap = new soapclient($WSDL-URL, //WSDL-Datei einlesen
2 array('login' => $user, 'password' => $password)); //Authentifizierung am
   Oracle UCM-Server
3
4 //Aufruf des Web Services
5 $ergebnis = $soap->CheckInUniversal(array(
6     'dDocAuthor'=>$user, //Autor des Bildes
7     'dDocTitle'=>'testBild4nagios', //Titel des Bildes
8     'dSecurityGroup'=>'private', //Sichtbarkeit des Bildes
```

```

9     'dDocAccount'=>'NAGIOS/TEST', //Angabe einer Gruppe
10    'dInDate'=>date("d.m.y H:i"), //Aktuelles Datum
11    'dDocType'=>'Picture', //Dokumententyp
12    'doFileCopy'=>'1', //Datei nur kopieren, nicht verschieben
13    'dDocFormat'=>'image/png', //MIME-Type
14    'primaryFile'=>array(
15        'fileName'=>'testBild4nagios',
16        'fileContent'=>$content) //Byteweise eingelesenes Bild
17 );
18 [...]
19 //Auswertung der Antwort des Servers
20 if (ereg(' erfolgreich eingecheckt.', $output)) {
21     echo('CHECKIN OK - '.$output);
22     die(0); //Einchecken erfolgreich
23 } else {
24     echo('CHECKIN CRITICAL - '.$output);
25     die(2); //Einchecken fehlgeschlagen
26 }

```

Listing 15: Aufruf des Web Services CheckInUniversal

Die Bilddatei muss für die Übertragung über **HTTP** zuerst byteweise eingelesen werden und anschließend mit dem Base64-Algorithmus kodiert werden. Dabei übernimmt die nuSOAP-Bibliothek die Base64-Enkodierung. Der Ablauf dieses Plugins soll durch das folgende Schaubild verdeutlicht werden.

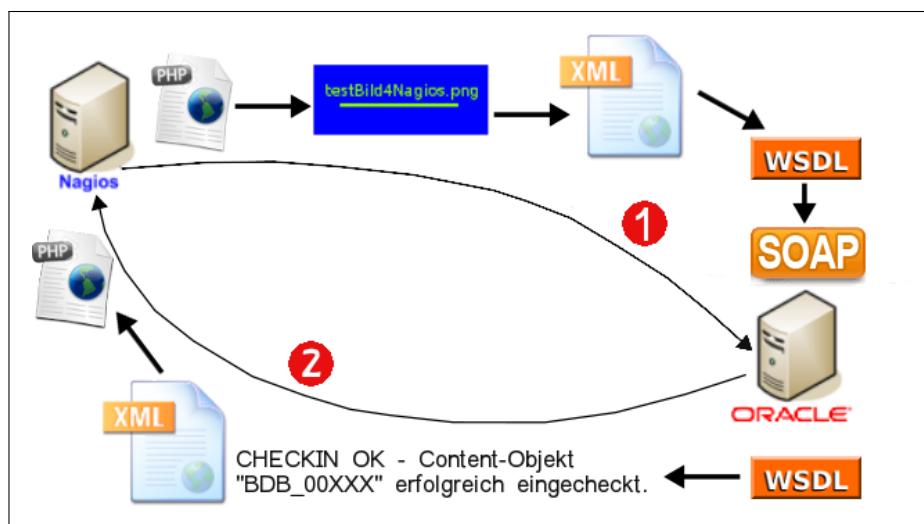


Abbildung 37: Einchecken eines Testbildes

1. Im ersten Schritt wird das **PHP**-Script vom Nagios-Server aufgerufen und liest das Testbild ein. Anschließend verwendet es die **WSDL**-Datei

des Web Services um das entsprechende **XML**-Dokument zu erstellen. Diese **XML**-Datei wird anhand der nuSOAP-Bibliothek **SOAP**-konform an den **Oracle UCM**-Servers gesendet.

2. Die **XML**-basierende Rückantwort des Servers wird auch anhand der **WSDL**-Datei erstellt und kann vom **PHP**-Script ausgewertet werden.

Validierung der Indizierung und Konvertierung Der zweite Teil der Benutzersimulation überprüft, ob das Testbild erfolgreich indiziert und konvertiert wurde. Dabei verwendet es die gleichen Grundfunktionen wie das erste Plugin. Jedoch wird anstatt dem Web Service *CheckInUniversal* die **WSDL**-Datei des Web Services *AdvancedSearch* verwendet und aufgerufen.

```
1 $ergebnis = $soap->AdvancedSearch(  
2     'queryText'=>"dDocTitle <substring> 'testBild4Nagios');
```

Listing 16: Überprüfen der Indizierung anhand einer Suchanfrage

Durch diesen Aufruf wird anhand seines Titels nach einem zuvor eingecheckten Testbild gesucht. Dadurch kann überprüft werden, ob das Bild korrekt vom Server angenommen und indiziert wurde. In der Rückantwort des Servers befindet sich unter anderem die eindeutige Identifikationsnummer des Testbildes. Diese Nummer wird für die Validierung der Konvertierung verwendet.

```
1 //Test des Originalbildes  
2 $ergebnisGet = $soap->GetFileByID('dID'=>$dID);  
3 [...]  
4 if(!mb_eregi('PNG', $outputGetOrig))  
5 {  
6     echo('SEARCH CRITICAL - Originalbild ist nicht im PNG Format!');  
7     die(2); //Originalbild korrupt  
8 }  
9 //Test der Thumbnailversion des Testbildes  
10 $ergebnisGetThumbnail = $soap->GetFileByID(array('dID'=>$dID, 'rendition' =>  
11     'Thumbnail'));  
12 [...]  
13 if(!mb_eregi('JFIF', $outputGetThumbnail))  
14 {  
15     echo('SEARCH CRITICAL - Thumbnailversion des Testbildes ist nicht im  
16     JPEG Format!');  
17     die(2); //Thumbnailversion korrupt  
18 }
```

Listing 17: Überprüfen der Indizierung anhand einer Suchanfrage

Sofern das Bild gefunden wurde, wird es vom Plugin anschließend angefordert und nach dem Dateityp untersucht. Für die Überprüfung auf den gültigen Datentyp wird direkt in den Binärkode der verschiedenen Versionen des Testbildes geschaut.

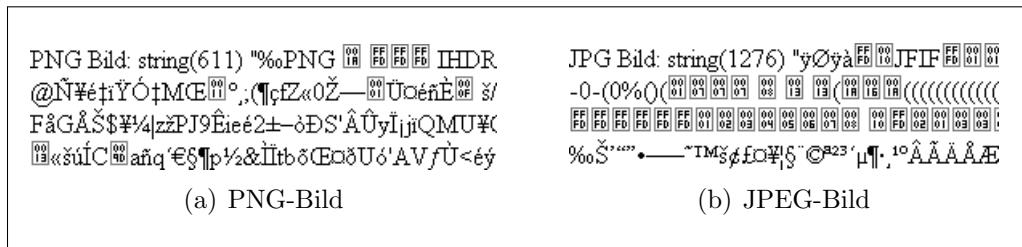


Abbildung 38: Binärkode von PNG- und JPEG-Bilddateien

Dabei wird für das Originalbild nach PNG und für die in JPEG konvertierten Versionen nach JFIF (JPEG File Interchange Format) geschaut.

Der Ablauf dieses Plugins ist dem ersten sehr ähnlich, wie Abbildung 39 zeigt.

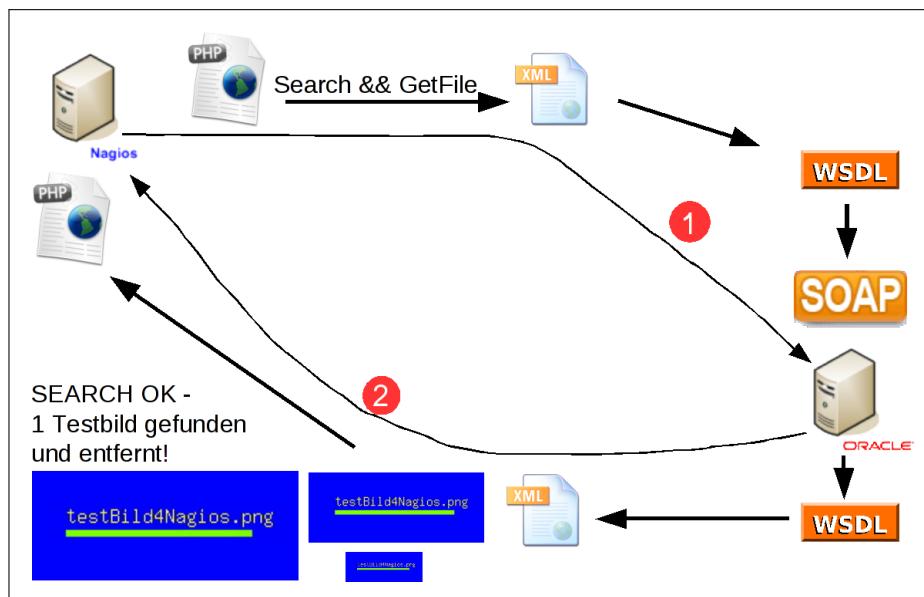


Abbildung 39: Validierung der Indizierung und Konvertierung

1. Zuerst wird die WSDL-Datei für den Web Service *AdvancedSearch* eingelesen. Eine Suchanfrage nach dem Testbild wird wieder über eine

XML-Datei an den Server gesendet. Wenn eine Datei gefunden wurde, wird das Originalbild und die konvertierten Versionen anhand der Identifikationsnummer angefordert.

2. Diese Bilder werden wieder byteweise innerhalb der **XML**-Datei der Serverantwort an den Nagios-Server übertragen und auf ihre Korrektheit überprüft. Sollten alle bisherigen Tests ohne Probleme abgelaufen sein, wird das Testbild und die konvertierten Bilder vom Server entfernt.

Damit nicht unnötiger Festplattenplatz durch die Testbilder verschwendet wird, sollen die Testbilder mit den konvertierten Version vom Server gelöscht werden.

Standardmäßig bietet **Oracle UCM** keinen Web Service zum Löschen von Dokumenten an. Daher muss zunächst eine **WSDL**-Datei dafür erstellt werden, siehe Abbildung 40.

Name	Typ	IdcName	Aktiviert
dID	field:int		true

Name	Typ	IdcName	Aktiviert
DOC_INFO isAllRevisionsDeleted	propertylist:CustomDocMeta field:boolean		true true

Abbildung 40: Anlegen eines eigenen Web Services

Der Name der **WSDL**-Datei und des Web Services kann beliebig gewählt werden, solange als Service die entsprechende interne Bezeichnung zum Löschen von Dokumenten „*DELETE_REV*“ verwendet wird.³⁷ Um Zweideutigkeiten

³⁷Quelle: [Huff06] S. 379

zu vermeiden, wird als Anforderungsparameter die eindeutige Identifikationsnummer verwendet.

Der eigene Web Service wird wie die anderen im Anschluss aufgerufen:

```
1 $ergebnisDelete = $soap->DeleteRevisionByID('dID'=>$dID);
```

Listing 18: Aufruf des eigenen Web Services

Die erstellten PHP-Dateien müssen noch Nagios als Befehl hinzugefügt werden.

```
1 #Einchecken eines Testbildes
2 define command{
3     command_name      check_ucm_checkin
4     command_line      /usr/lib/nagios/plugins/check_ucm/nagiosCheckin.php
5 }
6 #Validierung der Indizierung und Konvertierung
7 define command{
8     command_name      check_ucm_search
9     command_line      /usr/lib/nagios/plugins/check_ucm/nagiosSearch.php
10 }
```

Listing 19: Befehldefinitionen der Benutzersimulation

Die Aufteilung der Benutzersimulation in zwei Nagios-Kommandos sorgt dafür, dass es keine Garantie für die Reihenfolge der Ausführung der Plugins gibt. Aufgrund dieser Asynchronität und dem Umstand, dass die Indizierung und Konvertierung des Testbildes abhängig von der Auslastung des **Oracle UCM**-Servers ist, wird die Anzahl der Ausführungen des Plugins für den Wechsel von Soft zum Hard State erhöht.

```
1 define service{
2     use                  generic-service
3     host_name           example.kit.edu
4     service_description Oracle UCM Search Delete
5     max_check_attempts  8
6     check_command       check_ucm_search
7 }
```

Listing 20: Angepasste Servicedefinition für die Benutzersimulation

8 Ergebnis

In dieser Arbeit wurde eine Nagios-Umgebung entwickelt, die eine umfassende Überwachung von **Oracle UCM** ermöglicht. Das Ergebnis lässt sich im Webinterface von Nagios betrachten, siehe Abbildung 41.

Host ↗	Service ↗	Status ↗	Last Check ↗	Duration ↗	Attempt ↗	Status Information
bilddatenbank ka.fzk.de	BDB local user auth	OK	2009-07-22 15:17:48	9d 20h 27m 19s	1/4	HTTP OK HTTP/1.1 200 OK - 35543 bytes in 0.011 seconds
	C:\Drive Space	CRITICAL	2009-07-22 15:16:19	6d 6h 1m 20s	1/4	c - total: 19.99 Gb - used: 16.31 Gb (82%) - free 3.67 Gb (18%)
	CPU Load	OK	2009-07-22 15:13:48	4d 1h 15m 27s	1/4	CPU Load 5% (5 min average)
	CPU Load active	OK	2009-07-22 15:18:20	0d 1h 5m 24s	1/4	CPU OK - CPU0 = 0%
	D:\Drive Space	CRITICAL	2009-07-22 15:17:07	6d 6h 38m 57s	1/4	d - total: 20.00 Gb - used: 9.36 Gb (47%) - free 10.64 Gb (53%)
	HTTP	OK	2009-07-22 15:17:07	9d 20h 27m 13s	1/4	HTTP OK HTTP/1.1 200 OK - 35543 bytes in 0.088 seconds
	IDC Content Admin Service hdb admin	OK	2009-07-22 15:17:36	6d 6h 38m 47s	1/4	IdcAdminService hdb_admin: Started
	IDC Content Service hdb	OK	2009-07-22 15:14:14	6d 6h 38m 51s	1/4	IdcContentService hdb: Started
	IDC RefineryService idc	OK	2009-07-22 15:16:56	6d 5h 37m 44s	1/4	IdcRefineryService idc: Started
	IIS Admin Service	OK	2009-07-22 15:16:47	6d 6h 37m 54s	1/4	IISADMIN Started
	IIS Prozess	OK	2009-07-22 15:14:08	3d 16h 42m 6s	1/4	w3wp.exe: Running
	IdcAdminNT	OK	2009-07-22 15:16:20	3d 16h 47m 37s	1/4	IdcAdminNT.exe: Running
	IdcRefLog Check	OK	2009-07-22 15:17:32	0d 0h 1m 12s	1/1	OK - no errors or warnings
	IdcServerNT	OK	2009-07-22 15:17:36	6d 6h 30m 27s	1/4	IdcServerNT.exe: Running
	Memory Usage	CRITICAL	2009-07-22 15:14:27	6d 6h 8m 57s	1/4	Memory usage: total 12335.51 Mb - used: 1599.87 Mb (63%) - free: 935.64 Mb (37%)
	NSClient++ Version	OK	2009-07-22 15:14:35	6d 6h 36m 36s	1/4	NSClient++ 0.3.6.818 2009-06-14
	Oracle UCM Checkin	OK	2009-07-22 15:14:43	3d 16h 32m 28s	1/4	CHECKIN OK - Content-Objekt "BDB_009201" erfolgreich eingecheckt
	Oracle UCM Search	OK	2009-07-22 15:15:56	0d 0h 2m 48s	1/8	SEARCH OK - 1 Testbild gefunden und entfernt!
	Oracle UCM Session	OK	2009-07-22 15:15:01	0d 17h 13m 49s	1/4	Oracle OK - result:20 match:none
	OracleServiceXE	OK	2009-07-22 15:14:33	6d 6h 9m 1s	1/4	OracleServiceXE: Started
	OracleXEINSListener	OK	2009-07-22 15:15:33	6d 6h 39m 17s	1/4	OracleXEINSListener: Started
	PING	OK	2009-07-22 15:17:46	6d 6h 55m 0s	1/4	PING OK - Packet loss = 0%, RTA = 0.45 ms
	Symantec AntiVirus	OK	2009-07-22 15:17:44	6d 6h 30m 19s	1/4	Symantec AntiVirus: Started
	TNS Listener	OK	2009-07-22 15:14:43	3d 7h 48m 3s	1/4	OK (10 ms)
	Windows Uptime	OK	2009-07-22 15:17:41	1d 0h 41m 3s	1/4	System Uptime - 35 day(s) 2 hour(s) 21 minute(s)
	Zeitdienst	OK	2009-07-22 15:14:42	6d 6h 9m 6s	1/4	W32TIME: Started
	pagefile active	OK	2009-07-22 15:14:32	0d 17h 14m 12s	1/4	Page File Utilization OK - D:\pagefile.sys = S2%
	uname active	OK	2009-07-22 15:13:46	0d 21h 49m 58s	1/4	Operating System OK - Microsoft(R) Windows(R) Server 2003 Standard Edition Service Pack 2

Abbildung 41: Webinterface von Nagios

Die einzelnen Elemente der Überwachung lassen sich mit Ausnahme der Benutzersimulation durch Konfiguration von bereits vorhandenen Nagios-Plugins überprüfen. Bei dieser Konfiguration müssen die bereits im Kapitel 3.1 erörterten Gesichtspunkte wie Netzwerkabhängigkeiten, -belastung und Sicherheitstechnische Aspekte bedacht werden. Gerade bei der Verwendung von Benutzerinformationen zur Authentifizierung ist es notwendig zu überprüfen, ob die Informationen als Klartext oder verschlüsselt übertragen wer-

den. Diese Überprüfung lässt sich durch Programme, wie Wireshark³⁸, durchführen. Wireshark protokolliert alle ein- und ausgehenden Netzwerkverbindungen mit. Bei einer Ausführung eines Checks mit einem Nagios-Agenten, welcher keine Verschlüsselung ermöglicht, lassen sich Informationen abfangen.

Abbildung 42: Klartext im Mitschnitt

Währenddessen bei Agenten mit Verschlüsselung sich aus mitgeschnittenen Daten keine brauchbaren Informationen herauslesen lassen.

Abbildung 43: Verschlüsselte Übertragung von Informationen

Die durch die einzelnen Plugins gesammelten Informationen lassen sich zusätzlich im Detail aufrufen:

Service State Information	
Current Status:	OK (for 2d 17h 41m 56s)
Status Information:	CPU Load 2% (5 min average)
Performance Data:	'5 min avg Load'=2%;80;100;0;100
Current Attempt:	1/4 (HARD state)
Last Check Time:	2009-08-11 15:17:31
Check Type:	ACTIVE
Check Latency / Duration:	0.180 / 0.021 seconds
Next Scheduled Check:	2009-08-11 15:22:31
Last State Change:	2009-08-08 21:37:31
Last Notification:	N/A (notification 0)
Is This Service Flapping?	NO (0.00% state change)
In Scheduled Downtime?	NO
Last Update:	2009-08-11 15:19:20 (0d 0h 0m 7s ago)

Abbildung 44: Details der Prozessorauslastung

³⁸Quelle: <http://www.wireshark.org/>

Wie in Kapitel 4.2 erwähnt können zusätzliche Add-ons installiert werden, die von Nagios eingesammelten Informationen visualisieren. Dazu kann der verbreitete Nagios Grapher der Firma NETWAYS³⁹ verwendet werden. Dieser wertet die Performanzinformationen der Nagios-Plugins aus und erstellt davon Graphen.

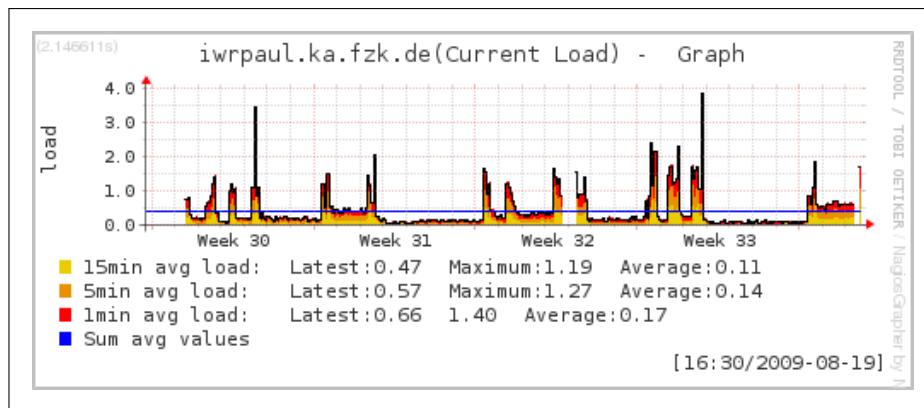


Abbildung 45: Darstellung der Prozessortauslastung

Die zeitliche Auflösung kann zwischen Minuten und Jahren dynamisch gewählt werden, um eventuelle Tendenzen zu erkennen. Die Anzeige der Arbeitsspeicherauslastung über einen längeren Zeitraum kann speicherintensive Programme zu entdecken.

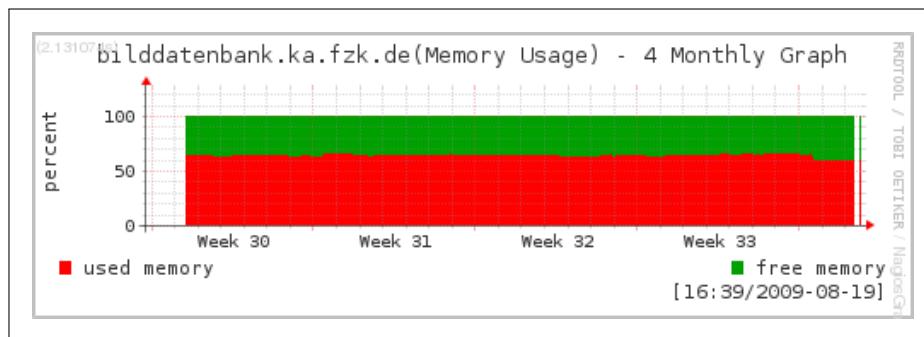


Abbildung 46: Arbeitsspeicherauslastung über vier Monate

³⁹Quelle: http://www.netways.de/de/produkte/nagios_addons/nagiosgrapher/

9 Zusammenfassung

Im Rahmen dieser Arbeit wurde eine Lösung entwickelt, um mit der Open Source-Überwachungssoftware Nagios den Betrieb des im Forschungszentrum Karlsruhe verwendeten Dokumenten-Management-Systems **Oracle UCM** zu überwachen. Eine solche Überwachung ist notwendig, um den Mitarbeitern des Forschungszentrums Karlsruhe einen möglichst zuverlässigen Dienst anbieten zu können. Dabei sollte die Überwachung proaktiv auf mögliche Fehlzustände testen und bei einer Störung eine Alarmmeldung an die verantwortlichen Kontaktpersonen versenden. Das Überwachungssystem sorgt dafür, dass jeder Fehler sofort gemeldet wird. Durch diese Meldungen können die Problemquellen vom Administrator gefunden und eventuell behoben werden, bevor die Endbenutzer Störungen bei der Nutzung des Dienstes bemerken.

Für die Bearbeitung der Aufgabe war es notwendig sich mit Grundlagen von Überwachungssystemen auseinanderzusetzen. Darunter fielen die Punkte Netzwerkstruktur, -abhängigkeit und verschiedene Sicherheitsaspekte, die beim Einsatz einer Überwachungssoftware eine Rolle spielen. Um die eigentliche Funktions- und Arbeitsweise eines Dokumenten-Management-Systems zu verstehen, wurde die grundsätzliche Art eines Dokumentes im Vergleich zu Daten betrachtet. Auf diesem Wissen aufbauend konnten die Aufgabenbereiche Eingabe, Verwaltung, Archivierung und Ausgabe eines Dokumenten-Management-Systems untersucht und verglichen zu Content-Management-Systemen gezogen werden.

Für die Umsetzung wurde die Service-orientierte Architektur der **Oracle UCM**-Anwendung in Verbindung mit Web Services verwendet. Dafür war das Verständnis der Grundprinzipien dieser Architekturen, deren Funktionsweise und von den Web Services verwendeten Elementen erforderlich. Dadurch konnte später korrekt auf die benötigten Funktionen zugegriffen werden.

Im Forschungszentrum Karlsruhe wird als Überwachungssoftware das Open Source-Programm Nagios für die Überwachung von Netzwerken, Servern und Diensten verwendet. Damit Störungen korrekt von Nagios erkannt werden, musste die Funktionsweise und der Aufbau dieser Software studiert werden. Die Informationen zur Auswertung werden durch Plugins gesammelt. Das Verständnis über die Struktur und Richtlinien dieser Plugins wurde benötigt, um später eigene zu entwickeln und sie effektiv zu verwenden. Dabei galt es, die speziellen Funktionen von Nagios wie die Hard- und Soft States oder das Flapping von Zustände bei der spätere Verwendung zu berücksichtigen. Über die verschiedenen Möglichkeiten von Nagios, um die benötigten Informationen zu sammeln, wurde ein kurzer Überblick gegeben.

Oracle UCM wird im Forschungszentrum Karlsruhe für die Verwaltung von Webseiten, Dokumenten und Bilder eingesetzt. Durch die Untersuchung des internen Aufbaus und der Arbeitsweise dieser Anwendung im Allgemeinen konnten die notwendigen Überwachungselemente ermittelt werden. Für diese Arbeit wurde der konkrete Einsatz von Oracle UCM als Bilddatenbank verwendet. Die dabei auftretenden typischen Benutzerinteraktionen wurden für die später folgende Benutzersimulation verwendet.

Die einzelnen Überwachungselemente wurden in die Ebenen Statusabfragen, Funktionalitätstest, Auswerten von Logdateien und Benutzersimulation unterteilt. Dabei führte die Abhängigkeit der Elemente zueinander zu der Einordnung in die verschiedenen Ebenen. Unter den Statusabfragen befinden sich einfache Tests wie ein Ping, Arbeitsspeicherauslastung oder der Zustand eines Prozesses. Bei den Funktionalitätstests werden die zu überwachenden Anwendungen durch Plugins überprüft und die Antwort ausgewertet, wie beispielsweise eine Anmeldung am Webserver mit Benutzerdaten. Die Benutzersimulation beinhaltet verschiedene Benutzeraktionen und überprüft, ob die Anwendung alle Funktionen erfüllt. Diese Einteilung in verschiedene Über-

wachungsebenen gibt den Verantwortlichen einen besseren Überblick über die Fehlersituation, so dass Fehlerquellen schneller entdeckt werden können.

Für die Umsetzung wurde ein Testsystem aufgesetzt, dass aus einer eigenen Nagios-Installation zum Testen der Überwachung und einer virtuellen Maschine, als Klon der Bilddatenbank, zum Simulieren der einzelnen Fehlzustände, bestand. Da es sich bei der zu überwachenden Bilddatenbank um einen Windows-Server handelte, wurde ein passender Nagios-Agent ausgewählt und dessen Installation und Konfiguration erläutert. Durch den Einsatz dieses Agenten konnten die verschiedenen Ebenen der Überwachung durch Verwendung von verschiedenen Überwachungsmethoden realisiert werden. Dabei wurde für die Benutzersimulation eigene Plugins entwickelt, die die Benutzeraktionen per Web Service ausführen. Das Plugin testet mit dem Hinzufügen eines Testbildes die Erreichbarkeit und einen Teil der Funktionalität des **Oracle UCM**-Servers. Durch eine Suchanfrage wird die Indizierung überprüft und die Konvertierung wird anhand der angeforderten Testbilder validiert. Dazu werden von den Plugins Web Services der **Oracle UCM**-Anwendung aufgerufen. Die Konsequenzen einer automatischen Benutzersimulation mussten beachtet werden. Durch die ständige Ausführung der Benutzersimulation würden die Ressourcen des Servers wie der Festplattenspeicherplatz an ihre Kapazitäten stoßen. Damit das Testbild und die konvertierten Versionen gelöscht werden konnten, musste ein Web Service in der **Oracle UCM**-Anwendung angelegt werden.

Alle Überwachungselemente sind im Webinterface vom Nagios aufgeführt und erlauben den Administratoren einen schnellen Überblick über die korrekte Funktion der Anwendung bzw. über die aufgetretenen Fehler. Die korrekte Benachrichtigung über Störungen konnte durch die Simulation der Fehlzustände in der virtuellen Maschine sichergestellt werden.

10 Ausblick

Bei einer Überwachung ist es wichtig, geeignete Schwellwerte zu setzen. An diesen Werten kann die Überwachungssoftware festlegen, ob ein Objekt einen kritischen Zustand erreicht hat oder nicht. Für die Ermittlung dieser Größen müssen die Werte der Überwachungselemente über einen längeren Zeitraum beobachtet und analysiert werden. Aufgrund des begrenzten zeitlichen Rahmens dieser Arbeit konnte dies nicht vollständig umgesetzt werden, so dass es während dem Betrieb fortgesetzt werden muss. Hierunter fallen vor allem spezifische Merkmale eines bestimmten Servers. Eine ungewöhnlich hohe Prozessorauslastung, die durch eine zeitlich gesteuerte Sicherung entstehen kann, sollte von der Überwachungssoftware nicht als Fehlverhalten interpretiert werden. Auch die Liste der Stopwörter für die Auswertung der Logdateien muss für neue, bisher unbekannte Fehler immer wieder erweitert werden. Das entwickelte Plugin für die Benutzersimulation kann mit zusätzlichen Funktionen versehen werden. Durch die Verwendung von anderen Web Services können weitere Funktionalitäten überprüft werden. Dabei kann die Benutzersimulation auf anderen Dokumenten-Management-Systemen eingesetzt werden. Das Plugin kann leicht angepasst werden, um anstatt eines Testbildes, beispielsweise den Check-In, Indizierung und Konvertierung einer PDF-Testdatei oder eines Word-Dokuments zu überwachen.



Glossar

Bezeichnung	Beschreibung	Seiten
AIIM	Association for Information and Image Management	24
CI	Coded Information	16, 19, 21
CMS	Content-Management-System	22–24
DMS	Dokumenten-Management-System	1, 14, 18–20, 22–24
DNS	Domain Name System	40
DoS	Denial of Service	14, 54
ECM	Enterprise-Content-Management	24
FQDN	Fully Qualified Domain Name	60
FTP	File Transfer Protokoll	38
HTTP	Hypertext Transfer Protocol	27, 70
IIS	Internet Information Service	52, 63
IP	Identifikation Protokoll	11, 14, 39, 55, 57, 60
LDAP	Lightweight Directory Access Protocol	40



Bezeichnung	Beschreibung	Seiten
MIB	Management Information Base	40, 41, 56
NCI	None-Coded Information	15, 19, 21
NRPE	Nagios Remote Plugin Executor	39, 54– 56, 59, 60, 62, 66
NSCA	Nagios Service Check Acceptor	41, 54, 56
OCR	Optical Character Recognition	19
Oracle UCM	Oracle Universal-Content-Management	1, 7, 43, 44, 46, 48, 49, 53, 54, 66– 69, 71, 73–75, 78–80
PDF	Portable Document Format	5, 19, 45, 81
PHP	PHP: Hypertext Preprocessor	69–71
RPC	Remote Procedure Call	61

Bezeichnung	Beschreibung	Seiten
SNMP	Simple Network Management Protocol	40, 41, 54, 56, 57, 59
SOA	Service-Oriented Architecture	9, 24– 26, 44
SOAP	Simple Object Access Protocol	27–29, 71
SSH	Secure Shell	38, 39, 53–56
SSL	Secure Sockets Layer	62, 65
UDDI	Universal Description, Discovery and Integration protocol	27–29
W3C	World Wide Web Consortium	26
WSDL	Web Services Description Language	27–29, 69–73
XML	Extensible Markup Language	27, 28, 69, 71, 73



Abbildungsverzeichnis

1	Zentralistische Bearbeitung	10
2	Ausgelagerte Bearbeitung	11
3	Zusätzliche Netzwerkabhängigkeit und Netzwerkbelastung . .	12
4	Anteil an strukturierten Informationen	16
5	Aufgabenbereiche eines Dokumenten-Management-Systems . .	18
6	Sichtweise CMS gegenüber DMS	22
7	„any-to-any“ Content-Management Konzept	23
8	Simple Software Architektur eines Webshops	25
9	Hinzugefügte Serviceorientierte Komponente	26
10	Kommunikationprotokoll SOAP	28
11	Ablauf einer Web Service-Benutzung	29
12	Plugins als separate Komponente	31
13	Anzeige des Servers im Webinterface von Nagios	32
14	Beispielhafte manuelle Ausführung eines Servicechecks . . .	33
15	Beispiel für den zeitlichen Verlauf durch vers. Zustände . .	35
16	Verlauf von sich schnell wechselnden Zuständen	36
17	Verschiedene Überwachungsmöglichkeiten von Nagios . . .	37
18	Ausführung eines netzwerkbasierenden Servicechecks . . .	38
19	Manuelle Ausführung eines Servicechecks über SSH	39
20	Aktive Checks mit NRPE	39
21	Struktur der Management Information Base	41
22	Passive Checks mit NSCA	42
23	Oracle UCM Architektur	43
24	Beispielhafter Einsatz eines Content Servers	44
25	Bilddatenbank als Anwendung	46
26	Überwachungselemente	52
27	Abfrage von Windows-Ressourcen durch <i>check_nt</i>	58



28	Zugriff auf den NSClient-Dienst durch check_nt	58
29	Konfiguration des NSClient++ während der Installation	60
30	Kommunikation zwischen Nagios und NSClient++	60
31	Aufruf eines NRPE-Kommandos	62
32	Aufruf des NSClient-Dienstes	62
33	Prozess- und Dienstüberwachung im Nagios-Webinterface	64
34	Ausgabe der betreffenden Zeile in der Logdatei	67
35	Geplanter Ablauf der Benutzersimulation	68
36	Anforderungsparameter für CheckInUniversal aus der WSDL- Datei	69
37	Einchecken eines Testbildes	70
38	Binärkode von PNG- und JPEG-Bilddateien	72
39	Validierung der Indizierung und Konvertierung	72
40	Anlegen eines eigenen Web Services	73
41	Webinterface von Nagios	75
42	Klartext im Mitschnitt	76
43	Verschlüsselte Übertragung von Informationen	76
44	Details der Prozessorauslastung	76
45	Darstellung der Prozessortauslastung	77
46	Arbeitsspeicherauslastung über vier Monate	77

Codelistingverzeichnis

1	Nagiosschema für Objektdefinitionen	32
2	Definition eines Hostobjektes	32
3	Verkürzte Definition eines Hostobjektes	32
4	Verkürzte Definition eines Hostobjektes	33
5	Beispielhafte Definition eines Servicechecks	34
6	NSClient++ Konfigurationsdatei	61
7	Servicedefinition des NSClient-Checks	62
8	Server spezifische Befehlsdefinition	63
9	Verkürzte Servicedefinition des NSClient-Checks	63
10	Prozess- und Service-Check Servicedefintionen	63
11	Überwachung der Festplatten- und Prozessorauslastung	64
12	Funktionalitätstest der Benutzeranmeldung	65
13	Auslesen der Verbindungen zur Datenbank	65
14	Konfigurationsdatei für <i>check_logfiles</i>	66
15	Aufruf des Web Services CheckInUniversal	69
16	Überprüfen der Indizierung anhand einer Suchanfrage	71
17	Überprüfen der Indizierung anhand einer Suchanfrage	71
18	Aufruf des eigenen Web Services	74
19	Befehldefinitionen der Benutzersimulation	74
20	Angepasste Servicedefinition für die Benutzersimulation	74



Tabellenverzeichnis

1	Rückgabewerte für Nagios-Plugins	34
2	Übersicht der verschiedenen Unix-Agenten	55
3	Übersicht der verschiedenen Windows-Agenten	57

Quellenverzeichnis

- [DMS08] Götzer; Schmale; Maier; Komke (2008) „*Dokumenten-Management - Informationen im Unternehmen effizient nutzen*“ 4. Auflage,
dpunkt.verlag GmbH Heidelberg, ISBN13: 978-3-89864-529-4,
Einsichtnahme: 25.06.2009
- [Barth08] Wolfgang Barth (2008) „*Nagios - System- und Netzwerk-Monitoring*“ 2. Auflage,
ISBN13: 978-3-937514-46-8, Einsichtnahme: 25.05.2009
- [Huff06] Brian Huff (2006) „*The Definitive Guide to Stellent Content Server Development*“,
ISBN13: 978-1-59059-684-5, Einsichtnahme: 25.05.2009
- [Jose07] David Josephsen (2007) „*Bulding a monitoring infrastructure with Nagios*“,
ISBN13: 0-132-23693-1, Einsichtnahme: 16.06.2009
- [SOA07] Hurwitz; Bloor; Baroudi; Kaufman; (2007) „*Service Oriented Architecture For Dummies*“,
ISBN13: 978-0-470-05435-2, Einsichtnahme: 29.07.2009
- [Melzer08] Ingo Melzer (2007) „*Service-orientierte Architekturen mit Web Services: Konzepte - Standards - Praxis*“,
ISBN13: 978-9-8274-1993-4, Einsichtnahme: 29.07.2009
- [Munin08] Gabriele Pohl und Michael Renner (2008) „*Munin - Graphisches Netzwerk- und System-Monitoring*“,
ISBN13: 978-3-937514-48-2, Einsichtnahme: 05.04.2009

- [Nagios] Ethan Galstad (2009) „*Nagios Enterprises*“,
Quelle: <http://nagios.com/>
Stand: 2009, Einsichtnahme: 14.07.2009
- [NagiosFAQ] Ethan Galstad (2009) „*What does Nagios mean?*“,
Quelle: <http://support.nagios.com/knowledgebase/faqs/index.php>
Stand: 02.06.2009, Einsichtnahme: 09.06.2009
- [UCM07] Ohne Verfasser (2007) „*Oracle Application Server Documentation Library - Oracle Content Management 10gR3*“,
Quelle: http://download-west.oracle.com/docs/cd/E10316_01/cs/cs_doc_10/documentation/integrator/getting_started_10en.pdf
Einsichtnahme: 16.06.2009
- [UCMlog09] Unbekannter Author „vramanat“ (2009) „*Universal Content Management 10gR3 - Content Server Log File Information*“,
Quelle: <http://www.oracle.com/technology/products/content-management/cdbs/loginfo.pdf>
Stand: 20.01.2009, Einsichtnahme: 05.06.2009
- [OraPress] Letty Ledbetter (2009) „*Oracle Press Release - Oracle Buys Stellent*“,
Quelle: http://www.oracle.com/corporate/press/2006_nov/stellent.html
Stand: 02.11.2006, Einsichtnahme: 16.06.2009

- [W3WS04] Booth; Haas; McCabe u.a. (2004) „*Web Services Architecture - W3C Working Group Note 11 February 2004*“,
Quelle: <http://www.w3.org/TR/ws-arch/wsa.pdf>
Stand: 11.02.2004, Einsichtnahme: 29.07.2009
- [NSClient] Michael Medin (2009) „*Using NSClient++ from nagios*“,
Quelle: <http://nsclient.org/nsSCP/wiki/doc/usage/nagios> (modifiziert)
Stand: 14.06.2009, Einsichtnahme: 12.07.2009
- [ClubOra] Unbekannter Author „Sadik“ (2008) „*Architecture of Oracle UCM*“,
Quelle: <http://www.club-oracle.com/forums>
Stand: 25.10.2008, Einsichtnahme: 12.07.2009
- [nuSOAP] Ahm Asaduzzaman (2003) „*Building XML Web Services with PHP NuSOAP*“,
Quelle: <http://www.devarticles.com/c/a/PHP/Building-XML-Web-Services-with-PHP-NuSOAP/1/>
Stand: 06.02.2003, Einsichtnahme: 12.08.2009