Dirk Quitschau

Planung und Entwurf eines Systemkerns für ein Netzwerkmanagementsystem zur Erstdiagnose

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Fachbereich Elektrotechnik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Thomas Schmidt
Zweitgutachter : Prof. Dr. rer. nat. Gunter Klemke
Abgegeben am 07. Juli 2005

Dirk Quitschau

Thema der Diplomarbeit

Planung und Entwurf eines Systemkerns für ein Netzwerkmanagementsystem zur Erstdiagnose

Stichworte

SNMP, VoIP, MIB, QoS, Thread Zustandsautomaten, Tomcat, JSP, Servlets, Struts, Validator-Framework, Tiles-Framework

Kurzzusammenfassung

Diese Arbeit befasst sich mit der Planung und dem Entwurf eines Netzwerk Management Systemkerns, mit dem es ermöglicht werden soll, Datenreihen auf Basis von SNMP-Abfragen von entfernten Systeme abzurufen und zu protokollieren. Diese Rohdaten können im weiteren Verlauf durch unterschiedliche Auswertungen bearbeitet und in Form von Diagramme zusammengefasst dargestellt werden.
Durch die Auswahl von verschiedenen Systemparametern unterschiedlicher Systeme, die gemeinsam durch Diagramme visuell dargestellt werden können, lassen sich u.U. durch einfache Korrelationsbildung Rückschlüsse bezüglich eines möglichen Ursprungs von Problemen und Phänomenen in Netzwerken ziehen.
Das Programm bietet die Möglichkeit, unterschiedliche Datensenken zu nutzen, die in XML konfiguriert werden können. Die View-Komponente ist exemplarisch in webbasierten Javatechniken (JSP, Servlets) implementiert worden.

Dirk Quitschau

Title of the paper

Planning and designing of a system core for a network management system assisting at first-level diagnostics

Keywords

SNMP, VoIP, MIB, QoS, thread state machines, Tomcat, JSP, Servlets, Struts, Validator-Framework, Tiles-Framework

Abstract

In this work we report on the concept and design of a network management core system. This lightweight approach allows for collecting and persisting SNMP data from distant systems, data processing and charting, with the aim of visualizing correlated performance data. These visualizations of relations and correlations between network parts enables the operator to quickly draw conclusions on trouble causes and easily detect irregular phenomena within the network. The core opens up the option to employ different databases, which can be freely configured by xml. The view components has been exemplarily implemented in web based java techniques (JSP, Servlets).

Inhalt

1. Einleitung

Aus heutiger Sicht bilden Netzwerke einen Großteil der Kommunikation in einem Unternehmen ab. Das Internet wird zur Informationsbeschaffung genutzt und dient häufig sogar als einzige Anbahnung von Geschäftsprozessen, wie z.B. bei einem Online-Shop. Allen kommunikativen Prozessen liegen Systeme zugrunde, deren Ausfall von wirtschaftlichen Verlusten und oft auch dem Verlust von Prestige begleitet wird.

Managementsysteme werden seit langem im Kontext mit aktiven Netzwerkkomponenten genutzt, da diese oftmals über Standards wie SNMP (simple network management protokol) und zusätzlich über propritäre Systemansteuerungen verfügen.

Die Telefonie über die Internetprotokollfamilie, kurz VoIP, nimmt ebenfalls immer größeren Stellenwert in der Kommunikationstechnik an. Allerdings findet sie ihre Anwendung weniger im privaten Bereich, sondern wird eher in unternehmenseigenen Netzwerken genutzt.
Die Vorteile einer preiswerteren Infrastruktur, die höhere Flexibilität im Betrieb paketvermittelnder Dienste und der unkomplizierten Integration in die heutige Bürokommunikation, haben sich viele Unternehmen schon überzeugen lassen VoIP einzuführen.

Mit den leitungsvermittelnden Telefontechniken prägten Qualitätsanforderungen, tritt automatisch auch ein äquivalenter Anspruch an die Qualität gegenüber den paketvermittelnden Techniken in Vordergrund. Es wird versucht mit dem Einsatz von QoS diesem Anspruch gerecht zu werden. Mit unterschiedlichen Ansätzen soll die Qualität von Punkt zu Punktverbindungen mit Qualitätsmerkmalen zu versehen werden, wie z.B. einer bevorzugten Behandlung vorrangiger Pakete in einer Routerqueue.

Bei allen Überlegungen, die die Kostenreduzierung in den Vordergrund stellen, soll hier auch auf die Gefahren hingewiesen werden, die eine Zusammenführung von Sprache und Daten auf einem gemeinsamen Medium mit sich bringt. Wurden vor einer Umstellung noch zwei getrennte Netze genutzt, die einander im Notfall teilweise ersetzen konnten, steht danach nur noch ein Netz zur Verfügung und es kommt damit zu einer gesteigerten Abhängigkeit von Diensten.
Die essentielle Frage bei Verwendung eines solchen Systems ist somit, wie einem Ausfall von Komponenten entgegenzuwirken ist. Oft steigt der Preis bei maximaler Ausfallsicherheit durch die Anschaffung zusätzlicher Hardware überproportional an, wie z.B. bei redundant ausgelegten

Anlagen. Eine Sekundäranlage überwacht die Funktionsweise der Primäranlage, um bei einer Störung Teilfunktionen oder die gesamte Funktion zu übernehmen.

Die Kosten könnten jedoch auch dadurch stark reduziert werden, dass anstelle von redundant ausgelegenem Anlangen die Wartungszeit bei Ausfällen möglichst gering gehalten wird. Ein Ansatz kann hierbei sein, das Wartungspersonal mit unterstützender Software bei der Fehlersuche auszurüsten, die eine geringere Einarbeitungszeit in das vorhandene Netzwerk gewährleistet und eine effektive Hilfestellung bei der Fehlereingrenzung bietet.
Ein System zu administrieren, welches auf IP Strukturen aufsetzt, erfordert meist einen Überblick über zusammenhängende Komponenten. Eine Fehlersuche bei paketvermittelnden Systemen beschränkt sich nicht nur auf die simple Verfolgung von Punkt-zu-Punkt-Verbindungen in einem zentral gesteuerten System. Vielmehr kommen die Problemstellungen von Verteilten Systemen zum Tragen und eine Fehlerquelle aufzuspüren wird komplex. Bei einer Diagnose liegt der Schwerpunkt beim komplexen Komponenten- und Ereigniszusammenspiel.
Ziel dieser Ausarbeitung soll ein System sein, welches einem Servicetechnikers bei der Erstdiagnose behilflich sein kann und Möglichkeiten bietet, Standardausfälle möglichst schnell zu entdecken und anschließend beheben zu können. Hierbei sind stetige Protokollierungsmöglichkeiten genau so wichtig, wie eine individuelle Datenerfassung von Systemparametern, die bei einer Langzeitüberwachung Aufschluss über sporadisch auftretende Fehlerquellen geben können.
Im weiteren Verlauf dieser Arbeit wird zunächst auf die Anforderungen eines Netzwerk Management Systems eingegangen und auf die wichtigsten Techniken, die im Zuge von Netzwerkmanagement standardisiert worden sind. Um die vordergründige Idee dieser Arbeit zu beschrieben werden der Ansatz der Korrelationsbildung von Datenreihen und das mögliche Zielumfeld des Systemkerns beschrieben werden.
Einen tieferen Einblick in den Systemkern CoVioN werden die Kapitel 5 und 6 gewähren, die darüber hinaus Aufschluss darüber geben werden, welche Funktionalitäten das Komponentendesign von CoVioN beinhalten wird. Auszüge der Implementierung sollen hierbei dem weiteren Verständnis dienlich sein.

2. Anforderungen an ein Netzwerkmanagementsystem zur Erstdiagnose vor Ort

Zunächst soll eine Möglichkeit geschaffen werden, um Daten von Systemparametern in regelmäßigen Abständen zu erfassen, um sie später in Graphiken, bzw. Diagramme darzustellen.

So können Zusammenhänge visuell erfasst werden und unter Korrelationsbetrachtungen zu Lösungsansätzen führen. Die Nutzdaten werden aus SNMP-Anfragen generiert, um im nächsten Schritt persisiert zu werden. Der Anwender erhält nun die Möglichkeit, die Daten unverändert in einem Diagramm darzustellen oder aber vorher mit verarbeitenden Auswertungen vorzuverarbeiten, um neben den Daten noch statistische Werte anzeigen zu lassen, wie z.B. die Werte einer kontinuierlichen Mittelwertbildung.
Komponenten, die in ihrer Funktion nicht zufriedenstellend arbeiten, würde man einer Langzeitüberwachung von Systemparametern unterziehen. Unterstützt durch graphische Darstellungen von Datendurchsätzen unterschiedlicher Interfaces wären so eventuelle Ressourcen-Engpässe oder burstartige Belastungen, durch simple Korrelationsbildung der Graphen erhebbar. Mit den Ergebnissen ließe sich eine Argumentationskette bilden, um bestimmte Phänomene in einem Netzwerk erklärbar zu machen. Zusätzlich dazu stellen die Erzeugnisse eine Hilfestellung dar, die bei Beseitigungen unterschiedlicher Problematiken zu Rate gezogen werden können. Der Techniker wird durch die Protokolle des Tools bei einer schriftlichen Ausarbeitung seines Servicefalles unterstützt.

Beispiel:

Die Mitarbeiter einer Firma beklagen sich über eine sehr langsame Verbindung zum Internet. Dies kann mehrere Ursachen haben. Zum einen besteht die Möglichkeit, dass die Router-Queues ständig voll sind, z.B. durch Viren oder Würmer im Intranet, oder aber die Verbindung zum Provider wäre mit hohen Paketverlusten belastet. Beide Fälle könnte man mit Hilfe von SNMP-Anfragen an den Router herausbekommen. Eine Möglichkeit könnte es hierbei sein, den Counter zu beobachten, der in der Mib-2 genau das Resetvorkommen von TCP mitzählt.

Wenn man von einem fest installierten System beim Kunden vor Ort ausgeht, hätte man durch eine clientseitige Konfiguration die Möglichkeit geschaffen, so genannte Traps zentral zu erfassen, um somit Zusammenhänge im Netzwerk besser nachvollziehen zu können.
Daten, die durch Auswahl von Systemparametern über eine bestimmte Zeit protokolliert werden, können auf unterschiedlichste Art weiter verarbeitet werden. Dabei ist es möglich, über eine simple Gegenüberstellung hinaus zu gehen. Nutzt man die mathematischen Vorgehensweisen der Korrelationsbildung, ließen sich hiermit Indizien erschließen, die einen Verdacht einer Verkettung von unterschiedlichen Systemphänomenen erhärten ließe.

Für den Korrelationskoeffizient gilt:

$$r = \frac{\sum_{i=0}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=0}^{n}(x_i - \bar{x})^2 (y_i - \bar{y})^2}} \quad wobei\ Mittelwert: \bar{x} = \frac{\sum_{i=0}^{n} x_i}{n}$$

Bei einem Vergleich von Datenreihen ließe sich für r eine Aussage darüber treffen, wie sehr Datenreihen einander korrelieren. Es gestaltet sich ein Wertebereich für r in der Form:

$$\forall r \in \mathbb{R} | -1 < r < 1$$

In der Interpretation wären Wertereihen, die sich dem Wert r=1 annähern, sich bedingende Systemparameter (wenn a steigt, steigt auch b). Im Gegenzug wären Werte, die sich gegen r=-1 annähern, ein Hinweis dafür, dass sich Systemparameter gegensätzlich verhalten (wenn a steigt, sinkt b und umgekehrt). Um eine graphische Übertragung einer Aussagekraft zu erreichen, ließe sich die Regressionsgrade in die Diagramme integrieren, für deren Ermittlung folgende Rechenvorschrift gilt:

$$y = rx + b \qquad r = Korrelationskoeffizient$$

übertragen auf die Sinnhaftigkeit einer Regessionsgraden:

$$b = \bar{y} - r\bar{x} \qquad \Rightarrow \qquad y = rx + (\bar{y} - r\bar{x})$$

Mit dieser Betrachtungsweise könnte erhält man die Möglichkeit Fehlerquellen besser in einem verteilen System entdecken zu können, wie folgendes Beispiel illustrieren soll:

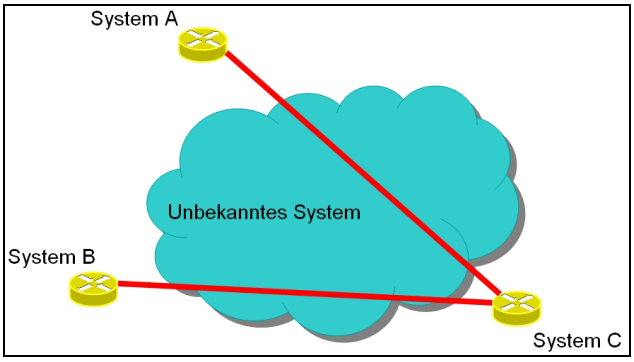

Abbildung 1: Beispiel zur Korrelationsbetrachtung

Entdeckt man beispielsweise in der Verbindung von System B nach System C, dass beim System B der Packetloss auf dem angeschlossen Interface stark zunimmt und beobachtet dabei, dass das selbe Phänomen zu der Zeit auch auf der Strecke zwischen System A und C auf dem Interfaces des Systems A stattfindet, lässt sich vermuten, dass der Fehler für diese ungewünschten Verhaltensweisen beim System C liegt. Eine Korrelationsbildung der Interfaces des Systems A und des Systems B könnte den Hinweis hierfür liefern.

3. Wesentliche Techniken und Standards in Netzwerk-Managementsystemen

3.1 SNMP

SNMP (Simple Network Management Protokoll) reiht sich in eine Reihe von Protokollen ein, die es schon lange gibt, jedoch nie an Lebendigkeit eingebüßt haben. Es wurde bereits 1980 in der Version 1 standardisiert und 1998 zu einer Version 2 weiterentwickelt. Ein weiterer Schritt in Richtung Sicherheit und Authentifizierung von Managementsystemen wurde mit der Version 3 im Dezember 2002 unternommen.
Die Unterschiede der einzelnen Versionen erstrecken sich von Erweiterungen in der MIB-Struktur, bis hin zu Änderungen bezüglich der Authentifizierung der Management-Applikationen an einem Agent. Wurde in der Version 1 die Authentifizierung noch über einen so genannten „shared key“ vorgenommen, der im Klartext über das Netz versendet werden musste, so wurde diese Sicherheitslücke schließlich in Version 3 durch diversen AAA Mechanismen (authentication, authorisation and accounting) endgültig geschlossen.

Die Architektur von SNMP ist ein Zusammenspiel von Netzwerk Management Systemen (NMS) und SNMP Agents. Hierbei wird die Kommunikation zwischen den Parteien mit SNMP-Messages vorgenommen. Die Graphik in Abb. 2 soll das Zusammenspiel kurz illustrieren:

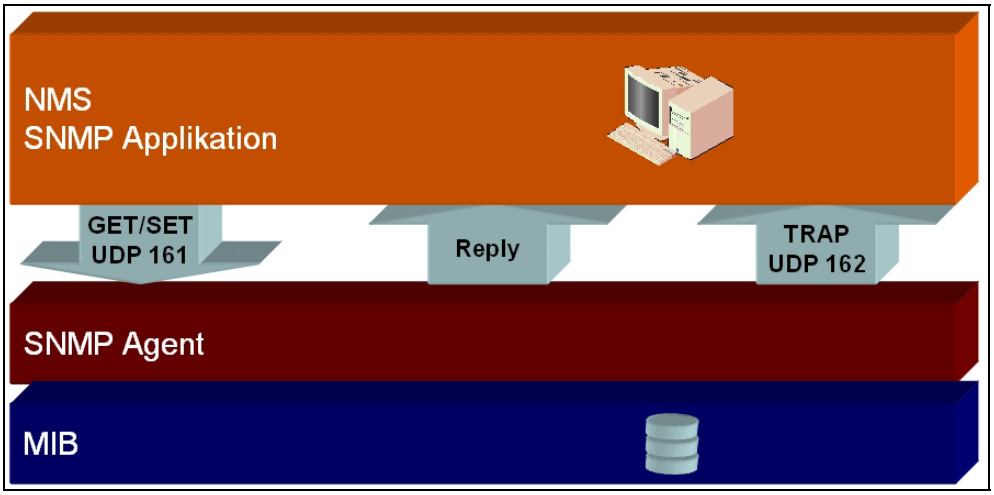

Abbildung 2: SNMP-Architektur

Die Kommunikation mit SNMP besteht aus SNMP-GET/-GET-NEXT, SNMP-SET, Reply und SNMP-TRAP Messages, die in UDP eingebettet werden.

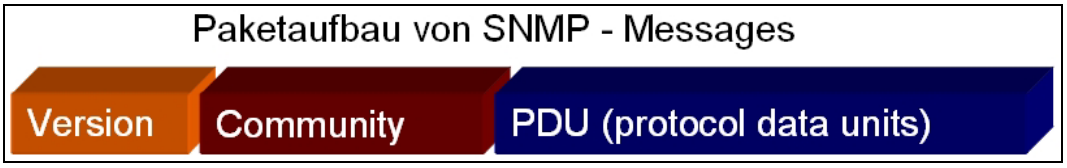

Abbildung 3: SNMP Paketaufbau

Eine SNMP-Message ist in drei Teile unterteilt:
Der verwendeten Versionsnummer, gefolgt vom community-String, der zur Authentifizierung von SNMPv1 und SNMPv2c verwendet wird, und so genannten protocol data units (PDU).
In den PDU's werden die unterschiedlichen Anfragen an den Agenten, dessen Antworten und die Traps kodiert.

An einem Beispiel soll kurz illustriert werden, wie beide Komponenten, der Agent und die NMS mit einander SNMP sprechen:

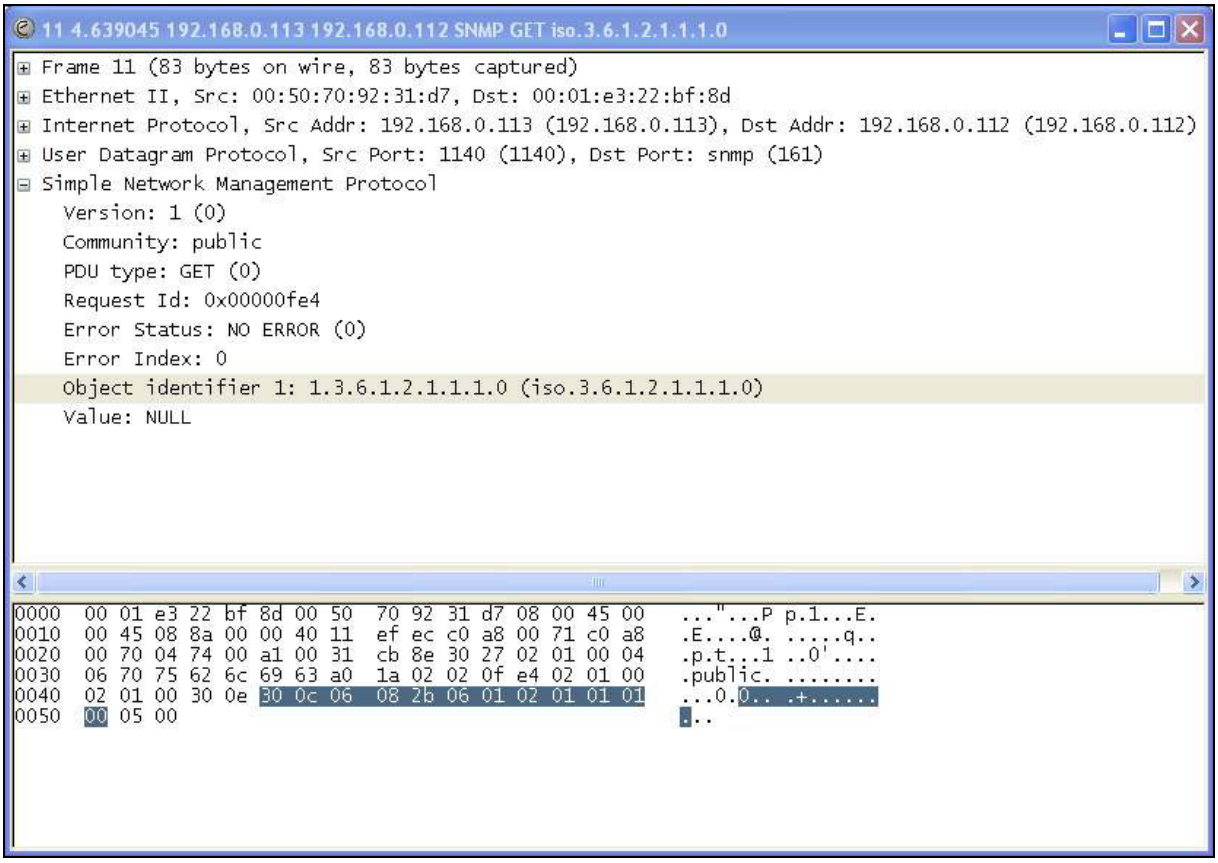

Abbildung 4: Darstellung eines SNMP-GET Request-Paketes mit dem Ethereal Packet Sniffer

In der oberen Abbildung 3 kann man erkennen wie das SNMP Protokoll aufgebaut ist. Als erstes finden wir die Version des Paketes (hier Version1), gefolgt vom Community String. Der angehängte PDU Teil splittet sich bei einer Anfrage auf in PDU Type (0=GET), einer Request-ID, einem Error-Status und Error-Index (beides bei Anfragen auf 0 gesetzt), dem Object-Identifier und einem Wert auf. Außerdem benutzt SNMP den UDP Port 161 für Requests, Responses hingegen unterliegen der Dynamik des Client-Ports (>1024). Die Management-Applikation besitzt die IP-Adresse 192.168.0.113, der Agent befindet sich auf einem Siemens Optipoint 410 mit der IP 114 desselben Subnetzes, einem VoIP-Telefon, welches die MIB-2 implementiert. Die Applikation stellt den SNMP-Request mit der OID 1.3.6.1.2.1.1.0 (Systembezeichnung) an das Telefon. Die Antwort sieht man auf der folgenden Abb. 5.

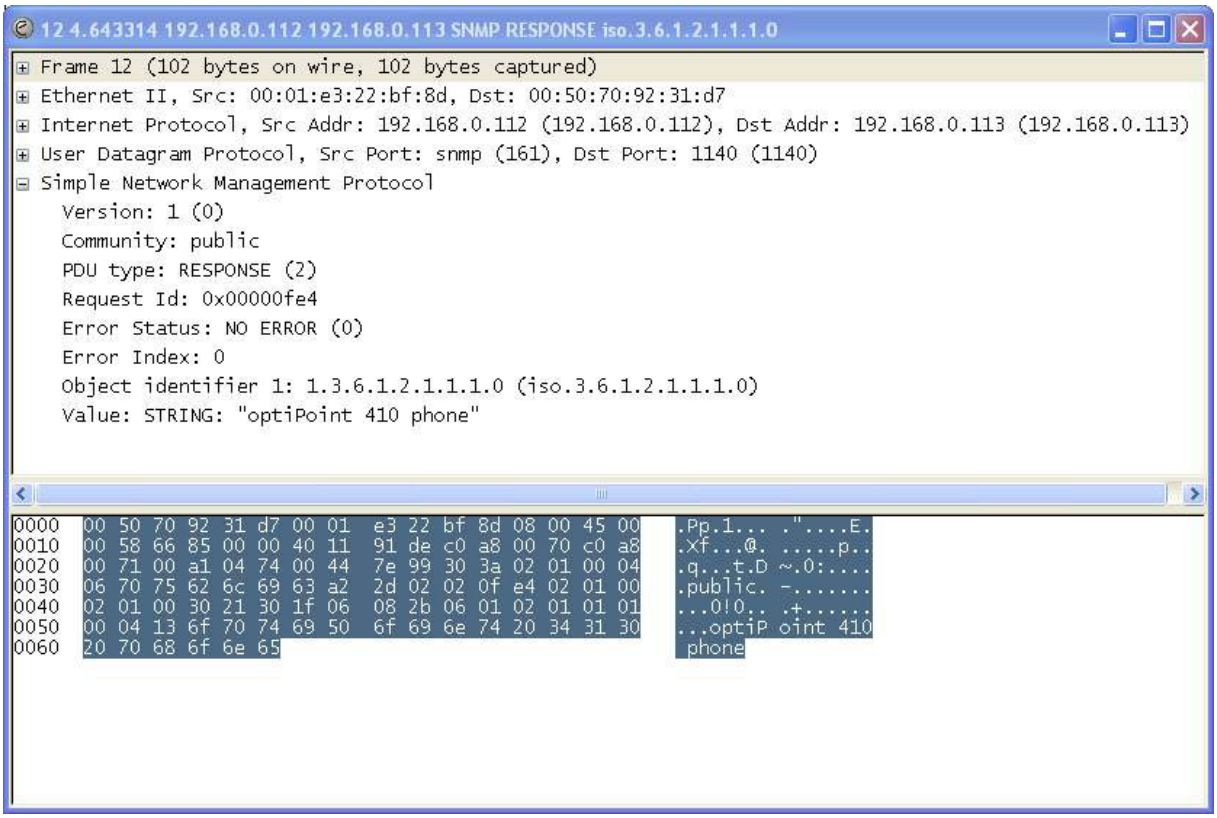

Abbildung 5: Darstellung eines SNMP Response-Paketes mit dem Ethereal Packet Sniffer

Die Antwort auf die SNMP Anfrage lautet „optiPoint 410 phone“. Wie man sieht stehen alle relevanten Daten in Klartext zur Verfügung. Vor der Paketstruktur änderte sich nur der PDU Type, dieser bekleidet den Wert 2 für die Markierung eines Response Paketes.

Es stehen mit dem SNMP Protokoll drei Schlüsseloperationen zur Verfügung:

GET und GET-NEXT

Ermöglicht der Managementapplikation eine Anfrage an den entfernten Agent zu stellen, wobei GET-NEXT zur Abfrage von Tabellen genutzt werden, deren Dimension der Managementapplikation nicht bekannt ist. Hier wird die OID der Tabelle gesendet und als Antwort erhält man den Wert der nächsten OID, die dem Agenten bekannt ist. Mit GET-NEXT besteht somit die Möglichkeit durch einen Baum oder Teilbaum zu iterieren.

SET

Ermöglicht der Managementapplikation einen Wert in der MIB Tabelle des Agents explizit zu verändern. Dies geht nur, wenn die Entität in der MIB als read-write deklariert wurde.

TRAP

Ermöglicht dem Agent einer vorher konfigurierten Management-Applikation Events zu senden, um über systemeigene Ereignisse ungefragt zu unterrichten. Traps sind grundsätzlich Bestandteil von MIBs, die gesondert aufgeführt werden. Es wird jedoch der Vorschlag in RFC 1215 [7] gemacht grundsätzlich coldStart(0), warmStart(1), linkDown(2), linkUp(3), authenticationFailure(4) und egbNeighborLoss(5) als Hersteller zu definieren.

Die Aufgabenverteilung von Agenten ist von MIB zu MIB unterschiedlich aufwendig. Die Agents bilden über eigene Systemaufrufe einen Informationsbaum, eine so genannte „MIB“ (Management Information Base), auf die im folgendem Punkt 3.2 näher eingegangen werden soll.
Die Informationen stellt es den Management-Applikationen zur Abfrage zur Verfügung. Die Agents haben die Aufgabe, ihre unterstützen MIB´s kontinuierlich mit Werten zu füllen und zusätzlich Traps an voreingestellte Netzwerk Management Systeme (NMS) zu senden, wenn bestimmte Ereignisse eintreten. Außerdem müssen sie Schreiboperationen, über SNMP-SET-Requests an ihrer Datenbasis verarbeiten können, was sehr umfangreich werden kann, da hier auch definierte Zugriffsrechte der MIB mit berücksichtigt werden müssen. Hinter den NMS befinden sich Applikationen, die diese Traps weiter verarbeiten. Bei der Datenerfassung seitens der Management Applikation hingegen, kann keine universell einsetzbare Lösung geboten werden. Jedes System hat ein individuelles Umfeld, Router unterschiedlicher Hersteller, Switche und andere Komponenten mit laufenden SNMP-Agenten. Somit muss ein Managementsystem immer von Einsatzgebiet zu Einsatzgebiet unterschiedlichen Anforderungsmerkmalen gerecht werden, bzw. dementsprechend konfigurierbar sein.

Da die für die Netzwerksicherheit von SNMPv1 (von 1988) wegen der in Klartext versendeten Community-Strings, immer ein Sicherheitsrisiko war, fand SNMP in dieser Version kaum seinen Einsatz in kommerziellen Netzwerken. Wegen dieser Schwäche von SNMP wurde 1993 von einer unabhängigen Gruppe zunächst das SMP (Simple Management Protokoll) entwickelt, welches nicht nur UDP und IP nutzte, sondern auch diverse OSI Protokolle und sogar Apple Talk unterstützte [28]. Die Entwicklung von SMP wurde von den Standardisierungsgremien von SNMP zum Anlass genommen, zwei Arbeitsgruppen ins Leben zu rufen. Eine war eher funktionsorientiert, die andere beschäftige sich mit der Umsetzung von sicherheitsspezifischen Themen. Ziel war es SNMP mit den Sicherheitsaspekten von SMP zu vereinigen, um so wieder einem genutzten Standard zu schaffen. Dies krönte schließlich mit dem SNMPv2c Standard, der

1996 in den RFC's 1901-1908 ([9] – [16]) verabschiedet wurde. Es wurden neue Datentypen verabschiedet, sowie weitere Access- Klauseln für Zugriffe auf Objectidentifier-Instanzen, wie accessible-for-notify (ein Objekt, welches nur zur Benachrichtigung beschreiben wird) und read-create (nicht nur verändern, sondern auch hinzufügen von Zeilen in einer Tabelle). Zudem wurden PDU-Types entwickelt, die allein der Management-Management Kommunikation dienen sollten. Der generelle Paketaufbau blieb derselbe. Mit dem PDU GET-BULK-Request wurde der Managementapplikation zusätzlich die Möglichkeit eingeräumt, eine vordefinierte Anzahl von Objekten gleichzeitig abzurufen. Hiermit gestaltete sich die Abfrage von ganzen Unterbäumen einfacher; wurde vorher für jedes Objekt ein eigener Request gesendet, konnten nunmehr Requests von ihrer Anzahl her und damit auch der SNMP-UDP-Verkehr minimiert werden.
Da die Version SNMPv2c bei der Handhabung der Sicherheit letztendlich dieselbe communitystringbehaftete Strategie gewählt hatte, wie ihr Vorgänger (daher auch das „c“ in SNMPv2c), wurde bereits im März 1997 eine neue IETF Gruppe gegründet, die die Sicherheitsmängeln von SNMP mit einem durchdachten Konzept begegnen wollte. Bis zum Januar 1998 hatte diese Gruppe mehrere vorgeschlagene Internet-Standards produziert, die als RFCs 2271-2275 ([17] – [21]) veröffentlicht wurden. In den RFCs wurden die Versionsunterschiede von Dispatcherinstanzen verarbeitet und gesteuert, die im Wesentlichen die alte communitystringbehaftete Startegie, als auch die neuen Sicherheitslösungen berücksichtigten. Die neuen Sicherheitsmechanismen umfassten MD5, SHA1, DES und HMAC (hash message authentication code).

Folgende Abb. 6 und Abb. 7 illustrieren den Aufbau von SNMPv3-Managern und SNMP-Agenten, die in RFC 3411 vorgeschlagen werden.

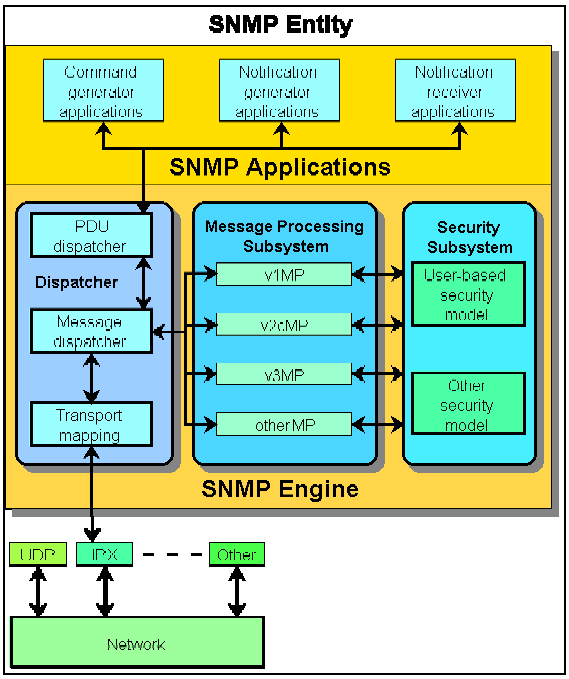

Abbildung 6: Achitektur eines SNMPv3 Managers

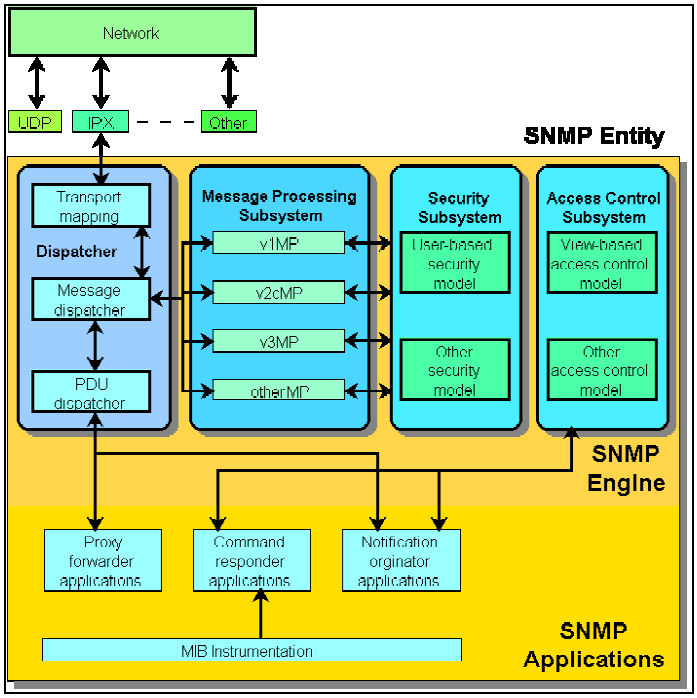

Abbildung 7: Architektur eines SNMPv3 Agenten

Nach diesem Leitbild implementiert Mibble SNMP in seinem Package.

3.2 Management Information Base (MIB)

Die Management Information Base stellt eine Konvention von unterstützten Systemparametern dar, die ein Agent für ein System ermittelt und per SNMP-Anfragen der Außenwelt zur Verfügung stellt. Sie besteht im einzelnen aus Elementen, die Objekte strukturiert in Baumform zusammenfasst. Für das SNMP-Protokoll fungiert sie wie eine Datenbank. Zur Beschreibung der Elemente wird ASN.1 (abstract syntax notation one) verwendet. Konstrukte und Datentypen sind in der SMI (structure of management information) standardisiert und festgelegt. Man verwendet ASN.1 bei der Beschreibung der MIB, um ein systemneutrales Instrument zu schaffen, welches von Softwarekomponenten zunächst interpretiert und verstanden werden muss, bevor man auf Systemobjekte über SNMP zugreifen kann. SMI ist in der RFC 1155 [5] beschrieben und umfasst

Regeln zur Erstellung von Mibs. Mit SNMP v2 wurde die SMI in RFC 1902 [10] zu SMIv2 weiter entwickelt und blieb hierbei abwärts kompatibel, indem die alten Datentypen erhalten blieben, jedoch z.B. Counter mit weiteren Datentypen versehen wurden, die ab Version 2 auch eine Ausprägung von 64 Bit haben können. Es gibt nur zwei unterschiedliche Strukturen von Datentypen; Scalare und zweidimensionale Tabellen, die wiederum Arrays von Scalaren darstellen. Die SMI hat auch die Baumstruktur vorgegeben, in der die Objekte oder Unterbäume angeordnet werden müssen. So definierte sie unter iso(1)-org(3)-dod(6)-internet(1) vier untergeordnete Nodes (directory-1 für jegliche Namensdienste, mgmt-2 für Menagementdienste, experimental-3 für experimentelle Nodes, um z.B. Standards, die sich noch in einem Entwicklungszustand befinden, vorübergehend einordnen zu können, und private-4 für herstellerspezifische Applikationen). Unter mgmt hatte vor der Mib-2 die Mib-1 ihre Position. Sie wurde vom IAB (Internet Architecture Board) in Zuge der Weiterentwicklung vollständig durch die Mib-II ersetzt. An dieser Stelle soll kurz darauf eingegangen werden, wie die Baumstruktur sich nach SMI in der MIB-II verändern darf:

- Durch vollständige Ersetzung eines Subtree's, wie es bei der Mib-1 durch Mib-2 geschehen war.
- Durch Erweiterungen der Blätter experimental und private
- Durch Ergänzung der MIB-II

ASN.1 beschreibt 28 unterschiedliche UNIVERSAL Typen, woraus SMI nur bestimmte Typen zur Beschreibung zulässt und zusätzlich mit eigener Nomenklatur versieht. Folgende Tabelle enthält eine Gegenüberstellung der Typen:

ASN.1	**SMI**
Universal 2	Integer
Universal 4	OctetString
Universal 5	null
Universal 6	ObjectIdentifier
Universal 16	Sequence, Sequence of

Abbildung 8: Typen Gegenüberstellung ASN.1 und SMI

In RFC 1155 wurde weitere Datentypen verabschiedet, darunter networkaddress, ipaddress, counter, gauge, timeticks und opaque.
Hierbei ist es von Agenten nicht zwingend notwendig, dass alle Werte der MIB mit Werten gefüllt werden. Sollten Einträge nicht vom Agenten unterstützt werden, darf er sie mit Null-Werten füllen. Außerdem sieht die Architektur von SMI vor, dass Agenten je nach Community-String, Teilbäume

sichtbar machen können oder READ / READ-WRITE Access damit verknüpfen - view based access. Hiermit können unterschiedliche Accessgruppen für jeden definierten Community-String definiert werden. Wie schon vorher erwähnt, sind MIB's grundsätzlich in einer Baumstruktur organisiert, wobei sich hinter jedem Knoten eine andere Auskunft über das System verbirgt. In der folgenden Abbildung wird ein Ausschnitt der MIB-2 dargestellt, die im RFC1213 (Request for Comments,[6]) als Standard verabschiedet wurde.

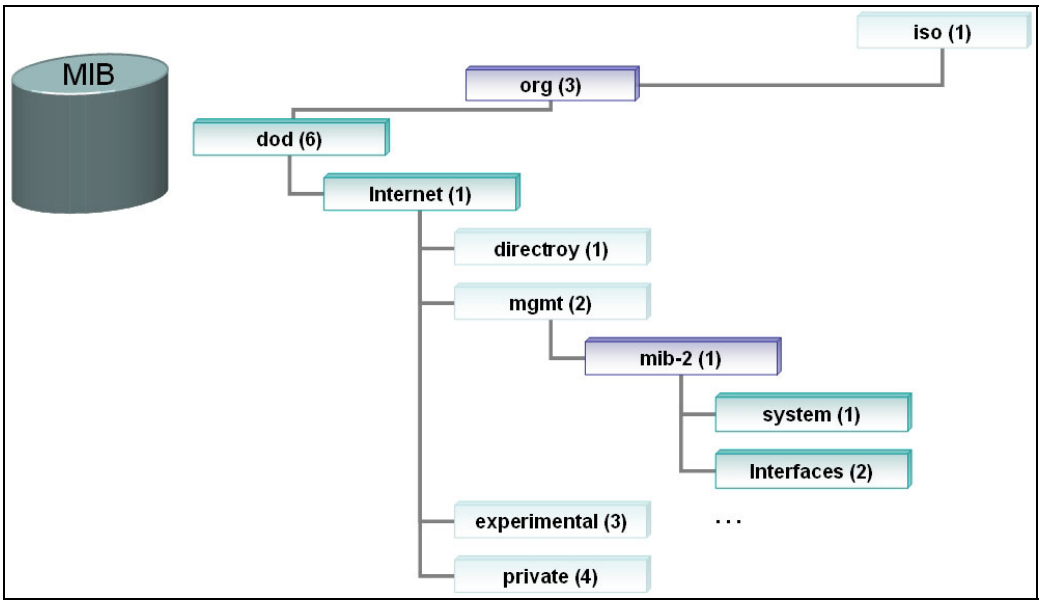

Abbildung 9: Abbildung der RFC1213 MIB-II

Soll beispielsweise die Systembeschreibung wie im vorangegangenen Punkt 3.1 über den Baum erreicht werden, so concatiniert man die Werte der benötigten Knoten, um an das gewünschte Blatt zu gelangen. Daraus erschließt sich eine OID (Objekt-Indentifier), die in einen SNMP-GET-Request gesetzt werden kann.
In dem gezeigten Beispiel erreicht man die Systembeschreibung über iso(1)=>org(3)=>dod(6)=>internet(1)=>mgmt(2)=>mib-2(1)=>system(1)=>SysDescr(1), also lautet die OID für die Systembeschreibung in diesem Fall „1.3.6.1.2.1.1.1“. Um die Instanz hinter der OID zu erhalten, ist es bei scalaren Werten so, dass als Instanz Identifizierer die Null konkatiniert wird. Somit ergibt sich der Vollständige Anfragestring „1.3.6.1.2.1.1.1.0“.
In der oberen Abbildung, der Darstellung der MIB-2, kann man schon am Aufbau der Struktur erkennen, dass sich auf Höhe der Baumtiefe 1.3.6.1.x unter anderem der Knoten private(4) befindet.
Unter diesem Node können Anbieter ihre eigenen MIBs für ihre Systeme veröffentlichen, um z.B. ganz spezielle Aspekte ihrer Hardware über SNMP zur Verfügung zu stellen.

3.3 Qualitäts- und Diagnoseparameter bei Voice over IP- Netzwerken

In einem VoIP-Umfeld sollte dem Wunsch nachgekommen werden, eine ständige Erreichbarkeit mit einer hohen Sprachqualität zu kombinieren.
Nur zu unterschiedlich sind die Arten, wie Daten in der paketvermittelnden oder bei PSTN übertragen werden. In der Paketvermittlung von IP gibt es mehrere Störfaktoren, die in der Natur des Protokolls liegen. So ist es z.B. so, dass für den Sprachcodec G.711 Pakete erzeugt werden müssen, die einer 8 Bit Datenbreite bei einer Abtastrate von 8 kHZ gerecht werden muss, wie es bei ISDN der Fall ist. IP kann nicht 8 Bit-Pakete mit 8 kHz versenden. Es werden mehrere Informationen zu größeren Paketen zusammengefasst um die Netzauslastung zu optimieren hinsichtlich Nutzdaten und Kontrolldaten. Zusätzlich kommt bei VoIP die Tatsache zum Tragen, dass auf beiden Seiten Codecs rechnen, um die Sprachdaten zu komprimieren (z.B. bei Sprechpausen). All dies trägt dazu bei, dass die Sprachqualität darunter leidet.
Die Vermittlung von IP-Paketen funktioniert zunächst unter dem „best effort"- Ansatz. Außerdem verlaufen aufeinander folgende IP-Pakete nicht zwingend entlang des gleichen Weges, um zum Ziel zu gelangen, so dass ein Endsystem auch mit der Sortierung der Paketreihenfolge beschäftigt wird, was ein Phänomen ist, welches nicht zwingend durch unterschiedliche Wegesuche zu belegen ist. Dies kann auch passieren, wenn Router in ihren Queues Pakete durcheinander bringen.
Im Gegensatz dazu haben wir in der PSTN-Technik fest geschaltete Verbindungen. Es müssen keine Bandbreiten oder Queues durchflossen werden; die Daten werden mit konstanter Bitrate und ohne spürbare Qualitätsverluste übermittelt.

Die Qualität von VoIP stellt Ansprüche an ein Netz, wie geringes Delays, geringer Jitter und geringen Packetloss, sowie dass Pakete in möglichst in chronologisch richtiger Reihenfolge beim Endsystem ankommt. Sonnst muss das Endsystem Pakete umsortieren (packet reordering).

Als Delay bezeichnet man die Verzögerungen der Signale in einem Netzwerk. Delays wirken sich, so lange sie gering sind nicht so sehr auf die Qualität der Sprache aus. Man veranschlagt pro gesprochene Silbe ungefähr 100 ms. Dies wird auch in VoIP-Systemen als oberstes Limit

erachtet. Bei einem wechselseitigen Gespräch jedoch hätten die Parteien bei zu hohem Delay eventuell das Problem, dass sie sich gegenseitig ständig ins Wort fallen würden.

Als Jitter bezeichnet man die Varianz der Delays, mit denen VoIP-Daten beim Empfänger ankommen. Wenn alle Pakete in einem Netzwerk die gleiche Verzögerung hätten, würde keiner eine Abnahme der Qualität verspüren, solange nur einer spricht. Wenn die ankommenden Pakete in ihrer Verzögerung stark variieren, schlägt sich dies sofort an der Softwarekomponente nieder, die die digitalen Daten in Echtzeit zurück in analoge Daten umwandeln werden müssen (z.B. den Ton zu einer Bildersequenz). In ihr existieren Buffermechanismen, die dem Jitterproblem entgegenwirken sollen, so genannte Jitterbuffer. Mit ihnen wird versucht unterschiedliche Delays in die Länge zu ziehen, um damit den Jitter zu kompensieren. Es ist also zunächst festzuhalten, dass Jitter direkten Einfluss auf die Sprachqualität ausüben, da ihre Glättung ein höheres Delay mit sich zieht und die Buffer nicht beliebig groß sein können, um sie zu kompensieren.

Wenn Pakete bei der Übermittlung verloren gehen, spricht man von Packetloss. Dieser sollte in einem VoIP-Netzwerk unter 1% liegen. Wie schon bei der Erklärung des Einflusses von Delays erwähnt, nimmt man pro gesprochen Silbe ein Δt von 100 ms an. Zu hoher Packetloss führt also direkt zu Verlusten von Sprachanteilen beim Abspielen der Gespräche – die Qualität von VoIP leidet imens unter Packetloss.

Endsysteme müssen, bevor sie Pakete verwerfen sicherstellen, dass sie in der Richtigen Reihenfolge verarbeitet werden. Kommen Pakete in nicht chronologischer Reihenfolge an, können auch hier bis zu einem gewissen Grad Buffermechanismen, sie wieder in die richtige Reihenfolge bringen, dies bezeichnet man Packetreordering am Endsystem. Treffen allerdings Pakete zu spät ein, werden sie verworfen. Da wir beim Packetreordering wieder Buffer genutzt werden, zieht dies wieder ein höheres Delay mit sich.

Die Wahl des Protokolls bei VoIP ist meist RTP (real time transport protcol). RTP wurde 2003 in RFC 3550 [25] von Schulzrinne et al in der Version 2 veröffentlicht und basiert auf der Vorgängerversion 1, welche in RFC 1889 [8] definiert ist. Das Protokoll setzt hierbei auf UDP auf und beinhaltet zeitliche Relationen zwischen allen Paketen mittels Timestamps. Neben dem Datentransport existiert RTCP, welches vor allem beim Aufbau und Abbau von Session-Streams genutzt wird, sowie dem Versand von Sender- und Receiver- Reports; eine Kommunikation mittels RTP funktioniert somit immer über zwei UDP-Ports, einen für den Datenfluss und einen für

die Kontrollkommunikation. In RTCP erhebt zudem noch statistisch verwertbare Informationen, die eben erwähnten Sender- oder Receiver-Reports (SR/RR). Diese verkehren in regelmäßigen Abständen zwischen den Gesprächsteilnehmern. Beide Reports beinhalten Daten, wie z.B. packetloss, sender octet counter, sender packet counter, interarrival jitter, Informationen, wann der letzte Report gesendet wurde und das Delay zwischen den Reports. Der Unterschied zwischen beiden Reports ist, dass der SR aus Sicht des aktiven Senders und die RR aus der Sicht des passiven Empfängers statistische Daten erhebt. Hierbei werden Heuristiken von Schulzrinne et al aufgestellt, wie oft Reports gesendet werden sollten. Ein durchschnittlicher Wert seien fünf Sekunden zwischen den Reports, der allerdings von Eckdaten abhängt, wie etwa Bandbreite, Anzahl der Teilnehmer an einem Stream und Ähnlichem.
Der Transport wird über RTP abgewickelt, welches in seiner Beschaffenheit über einen fixen Header verfügt und je nach medialer Anwendung mit zusätzlichen Headern, sog. Payloads, erweitert wird.

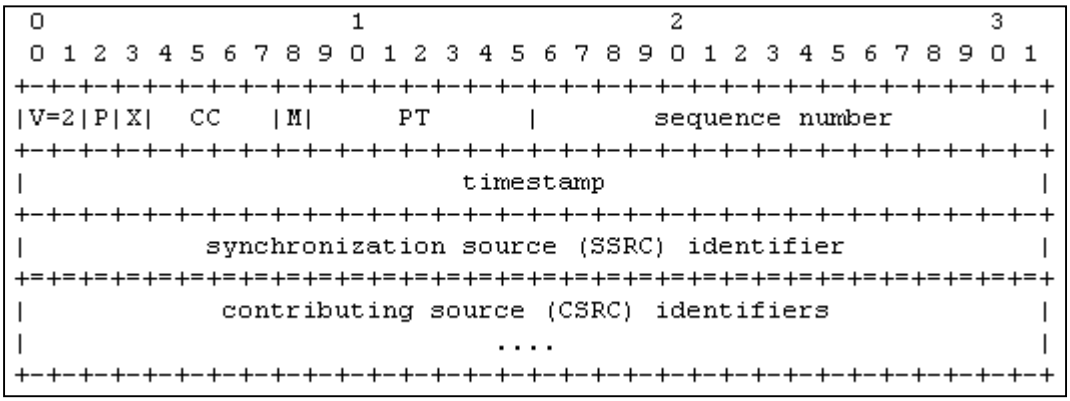

Abbildung 10: Fixed Header von RTP aus RFC 3550

Beschreibung des Headers:

V	Version
P	Padding Bit (für Headererweiterungen bei Verschlüsselungsalgoritmen, die eine fixe Headergröße benötigen)
X	Extensionbit (Zeichen dafür, dass dem fixed Header eine Medienerweiterung folgt)
CC	Zähler, wie viele CSRC's am Ende des Paketes angehängt sind
M	Markerbit
PT	Payloadtype (Art der Applikation, für die der Stream bestimmt ist. Standardmappings findet man in RFC 3551 [26]. Applicationen, die den payloadtyp nicht verstehen sollen diese Pakete nicht verarbeiten)
sequence number	sequencen number des Paketes (um direkt packetloss festzustellen)
timestamp	RTP Zeitstempel (nicht im allgemeinen Verständnis, wie ein Zeitstempel zu verstehen. Mehere Pakete können den selben Zeitstempel haben; es geht um die Abspielsynchronisation auf Empfängerseite, Bilder sollen z.B. mit dem dazugehörigen Ton gleichzeitig abgespielt werden)
SSRC	Identifiziert eine Streamquelle eindeutig
CSRC	Indentifiziert max. 15 streambeisteuernder Quellen; werden durch Mixer hervorgerufen; z.B. würden zwei microphone für Stereoaufnahmen jeweils eine CSRC zugeordnet werden

Abbildung 11: Beschreibung der Headerfelder des fixed RTP Headers

Für eine qualitativ zufrieden stellende Sprachübermittlung über IP werden besondere Ansprüche an Router, die für die Vermittlung der Pakete zuständig sind, gestellt. Sie müssen versuchen, VoIP-Pakete einem besonderen Vorzug bei der Vermittlung zukommen zu lassen. Die besondere Behandlung von VoIP-Paketen kann z.B. in der Router-Queue stattfinden.
In diesem Zusammenhang begegnet man Qualitätsansprüchen im IP-Umfeld mit Quality of Service (QoS), welches unter anderem eine Bandbreitensicherung beinhaltet.

Auch heute noch begegnet man dem Problem QoS mit einer Überdimensionierung der aktiven Netzwerkkomponenten, wie z.B. den Routerressourcen Bandbreite, Puffergrößen und verschiedenen Algorithmen zur Queueverarbeitung. Man erhofft sich somit jeder Art von Traffic gewachsen zu sein, scheitert jedoch u.U. bei burstartigem IP-Verkehr.
Hierbei ist UDP als zustandloses Protokoll für burstartigen Verkehr verantwortlich, da sich dagegen TCP, als höflich geltendes Protokoll, bei zu hohem Packetloss in den „slow start"-Zustand zurückzieht.
Quality of Service findet heute auf unterschiedlicher Basis seinen Einsatz. In einer Broadcastdomäne kann QoS auf Layer 2 abgebildet werden. Der 802.1p Standard stellt bis zu acht verschiedene Verkehrsklassen für fast alle 802.x-Protokolle bereit. Die Signalisierung der Benutzerpriorität kann auf zwei Arten geschehen:

- Durch das Priority-Feld in einigen MAC-Protokollen (z.B. 802.5, FDDI, 802.12 Demand Priority)

- Über die User Priority im IEEE 802.1Q Tag Header

Zur Übersetzung von Layer 2 Prioritäten in Layer 3 Prioritäten schlägt Niclas Ek von der Universität Helsinki folgende Einteilung von Nutzerklassifizierungen vor.

Ek Niclas [1]:

a) Netzwerk Kontrolle, sehr wichtig, um die Infrastruktur beizubehalten
b) Sprache mit weniger als 10 ms Delay
c) Video mit weniger als 100 ms Delay
d) controled load, einer wichtigen Applikation
e) excellent effort, für wichtige Anwender
f) best effort, gewöhnliche LAN Priorotät
g) Hintergrund auszuführen (Dateitransfer, Spiele, usw.)

Hierbei muss unterschieden werden, in welcher Form die Datenklassen zusammengefasst werden sollten, wenn beispielsweise von der aktiven Netzwerkkomponente nicht sieben Queues vorhanden sind. Folgende Zusammenfassung wird von Ek vorgeschlagen:

Vorhandene Queues	Zusammenfassung von Klassen
1	Alle in eine
2	a-d und e-g
3	a-b und c-d und e-g
4	a-b und c-d und e-f und g
5	a-b, c, d, e-f, g
6	a-b, c, d, e, f, g
7	a, b, c, d, e, f, g

Abbildung 12: Vorschlag von EK - Zusammenfassung von Queueklassen

Unterschiedliche Queues würden dann weiterhin mit Paketen „class based“ gefüllt. Für die Weiterverarbeitung auf den Routern könnte man ein unterschiedliches Scheduling der Queues vornehmen. Würden sie Queues einer bestimmten Bandbreite zugeordnet und die Pakete würden anhand dieser weiter vermittelt werden, würde man von „weighted fair queueing“ sprechen. Alternativ könnten zunächst die höher priorisierten Queues abgearbeitet werden, dies entspräche „class based queueing“. Denkbar wäre ebenfalls eine Mischung beider Versionen, was dem „weighted round robin“-Scheduling gleich kommen würde.
Ein weiterer wichtiger Gesichtspunkt ist das Verwerfen von Paketen, da je nach Anwendung unterschieden werden muss, wie wichtig die schon empfangenen Daten sind oder ob ein Verwerfen von neuen Paketen sinnvoller sein kann, als das Verwerfen von alten. Bei einer

Übertragung von Daten mittels FTP ist es sinnvoller alte Daten zu behalten, da die bestehenden Retransmit-Buffer sonst komplett verworfen werden müssten. Bei einer Kommunikation von Echt-Zeit-Daten oder Sprache hingegen sind alte Pakete nicht so wertvoll, wie erst kürzlich empfangene, evtl. kann sogar auf einige Pakete verzichtet werden. Wichtig bei einer Echtzeit-Datenverarbeitung ist, dass die chronologische Reihenfolge gewährleistet ist.
Es existieren neben dem eben genannten noch weitere Architekturen von QoS. Hierbei wurde 1998 von der IETF ein Layer 3 Architekturentwurf gefertigt, bei dem von IntServ- oder DiffServ-Ansätzen gesprochen wurde. Der IntServ- Ansatz (integrated service architecture) verfolgt hierbei die Idee Ressourcen für komplette Flüsse entlang des gesamten Weges zu reservieren. Dabei wirde oft vom deeply QoS-Ansatz gesprochen, weil die Bandbreite für die komplette Übermittlung des Datenflusses garantiert wurde. In diesem Zusammenhang wurde das Ressource reSerVation Protokol (kurz: RSVP) entwickelt, welches nicht nur für Unicast, sondern auch für Mulicast QoS Routerstates entlang eines Pfades etabliert. Dieser Ansatz verspricht die sauberste Interpretation von QoS darzustellen, jedoch entdeckt man bei genauerer Betrachtung einige gravierende Nachteile. So müssen alle beteiligten Komponenten, von der ursprünglichen Applikation bis hin zu jedem Router, durch den der Fluss geleitet wird, die RSVP-Messages berücksichtigen, d.h. interne Stati anlegen und die Pakete interpretieren, was einen erheblichen Overhead mit sich führt. Wegen der Notwendigkeit der intern geführten Stati, muss auf jeder verarbeitenden Einheit genügend Speicher reserviert und periodisch aktualisiert werden - somit skaliert eine Anwendung von RSVP nicht zufrieden stellend. Der IntServ-Ansatz instanziiert states per flow, und ist somit eine nicht vorhersagbare Größe.
In diesem Zusammenhang soll die Definition eines Flusses als eine unidirektionale Verbindung in Erinnerung gerufen werden, die durch ein 5-Tuple (Source IP, Source- Port, Destination IP, Destination-Port, Transportprotokoll) in IPv4 gekennzeichnet ist. In IPv6 markiert das Flow-Label, ob ein Paket zu einem Datenfluss gehört.
Der DiffServ- Ansatz (differentiated service architecture) dagegen versucht gar nicht erst komplette Wege mit Bandbreitenreservierungen zu belegen. Es definiert states nur per class, was bei einer fest definierten Anzahl von Klassen eine vorhersagbare Größe darstellt. Der DiffServ-Ansatz skaliert somit besser. Es wird versucht eine ganzheitliche Lösungsfindung auf kleine lokale Entscheidungen runter zu brechen. Dazu ist eine paketabhängige Klassifizierung an Hand der Internetprotokoll-Pakte vorgenommen worden. Hierzu herangezogen werden im Falle von IPv4 das TOS- Feld (type of service), bei IPv6 das Traffic-Class-Oktett. In RFC 2474+ [22] wird das Oktett aufgeteilt und ein Teil als DSCP (Differentiated services codepoint) beschrieben; dabei blieben die untersten zwei Bit ohne Funktion und die oberen sechs Bit boten somit 2^6

Serviceunterscheidungen. Wie die einzelnen Klassen von einem Router behandelt wurden, Queueing, Scheduling, Policing oder sharping, wurde als PHB (per hop behavior) betitelt. Alle Pakete die dem selben PHB zugeordnet werden, werden als BA (bevavior aggrgate) bezeichnet. Dabei wurde `000000` als *default PHB* definiert, was der best effort Vermittlung gleich kommt. `xxx000` ist als class-selector PHB definiert worden. Man beachte, dass der default PHB eine Untermenge von class-selector PHB darstellt. Beispielsweise wird ein Paket mit `110000` in Scheduling, Queueing usw. bevorzugt behandelt gegenüber einem Paket mit `100000`. Darüber hinaus besteht noch ein weiteres PHB, nämlich das EF PHB (expedited forwarding). Er soll den Router veranlassen Pakete mit EF PHB generell mit möglichst wenig Packetloss, wenig Delay, wenig Jitter und einer garantierten Bandbreite zu vermitteln. Das dazu notwendige DSCP lautet `1010110` und wird in RFC 2474 als solches festgelegt.
Im RFC 2597 [23] wird eine Erweiterung der PHBs vorgenommen. Das von RFC definierte Assured Forwarding (AFxy) stellt Regeln für das Verwerfen von Paketen (drop precedence) auf Grundlage von einer Codepointtable auf.

Verwerfungspriorität (Drop Precedence)	Klasse 1 (AF1)	Kalsse 2 (AF2)	Klasse 3 (AF3)	Klasse 4 (AF4)
Niedrige Dropquote	001 010	010 010	011 010	100 010
Mittlere Dropquote	001 100	010 100	011 100	100 100
Hohe Dropquote	001 110	010 110	011 110	100 110

Abbildung 13: Darstellung der Drop-KIssifizierungen nach RFC 2597

Auf den Routern konnte nunmehr für jede AF-Klasse ein bestimmtes Limit an Bandbreite konfigurierbar gemacht werden, um die oben angeführte Tabelle umzusetzen.

Um die Wichtigkeit von QoS in Bezug auf VoIP noch einmal zu unterstreichen noch einmal ein Beispiel:

Gehen wir davon aus, dass ein Weg, der von einem VoIP- Paket genommen wird über drei Hops geht, d.h. dass das Paket drei Router passieren muss, um seinen Endpunkt zu erreichen. Jeder Router garantiert dabei zu 97%, dass die behandelten Pakete nach deren individuellen Wünschen bearbeitet werden können. So ergibt sich eine Kette von prozentualen Werten, an deren Ende ein Wert steht, der vielleicht nicht mehr einem qualitativ ausreichenden Leistungsmerkmal entspricht.

Erreichbarkeit = 0,97*0,97*0,97 = 0,912673

Wie man sieht, würden auf diesem Weg fast 10% der Pakete nicht rechtzeitig vermittelt werden können. Dies ist ein Phänomen, welches oftmals nicht als Störquelle für eine schlechte Kommunikation erkannt wird, da zunächst meist im lokalen Areal nach Quellen gesucht wird. Es kann also sein, dass, nur weil ein Provider seine Router wartet und seinen Verkehr zeitweise umleitet, Störungen in der VoIP-Paketvermittlung auftreten, weil andere Provider wohlmöglich kein QoS oder Service Level Agreements (SLA) haben. Hier lässt es sich schon erkennen, warum es manche Unternehmen immer noch als ein Wagnis ansehen, ihre Kommunikationswelt komplett der IP-Welt anzuvertrauen. Sobald Pakete unterschiedliche Hoheiten von Netzwerkbetreibern durchwandern müssen, lässt sich lediglich noch feststellen, dass die Verbindung schlecht ist; eine Verfolgung der Pakete durch das Netzwerk ist nicht ohne weiteres Zutun möglich.

4. Ein anschauliches Netzwerkmanagement zur Unterstützung einer vor Ort Diagnose

Wartungstechniker haben ständig mit wechselnden Arbeitsumfeldern zu tun. Bei jedem Einsatz muss sich neu in ein Netzwerk zurechtgefunden werden und oftmals erhält man nicht die Möglichkeiten alle Netzwerkkomponenten auf deren Konfigurationsstand und Funktionsweise zu überprüfen.
Aus dieser täglichen Situation heraus wünschte sich ein Siemenstechniker ein leicht zu handhabendes Tool, mit dem er einfache Datendurchsätze graphisch ermitteln kann. Bei Nachforschungen nach bereits vorhandenen Lösungen fand sich z.B. der MRTG (Multi Route Traffic Grapher). Die Probleme der meisten frei erhältlichen Tools sind jedoch entweder eine sehr umständliche Handhabung oder aber eine uneinheitliche Programmiersprache. Für die Nutzung von MRTG definiert man sich statische Sichten und erhält für jede definierte Komponente bestimmte Auszüge aus einer MIB als Graphik in Form von HTML und PNG-Graphiken präsentiert. Die Konfiguration entsteht bei MRTG über Textfiles und CRON-Jobs.
Leider reicht die statische Sichtweise nicht aus, um komfortabel und spontan mehrere Komponenten auf unterschiedlichste Art und Weise zu untersuchen, Vergleiche anzustreben und wohlmöglich datenverändernde Operationen, wie eine kontinuierliche Mittelwertbildung, auf den Daten auszuführen. Entscheidet man sich für andere frei erhältliche Tools, so steht man auch oft einem System gegenüber, welches unterschiedliche Sprachen für spezielle Anforderungen einsetzt. Hier findet man beispielsweise die Sprache PEARL bei der Interaktion mit SNMP wieder,

oder CRON-Jobs, die periodische Arbeiten für eine Applikation erledigt. Das Zusammenspiel der Komponenten birgt dies Gefahren bei Versionswechseln einzelner Komponenten.
Daher fiel die Entscheidung ein neues Tools mit einer einheitlichen Sprache zu erschaffen. Hier fiel die Wahl auf Java, da diese Programmiersprache international anerkannt und genutzt wird und sich alle Schattierungen von Anwendungsbereichen sich in ihr wieder spiegeln. Eine uniforme Architektur der verwendeten Komponenten stellt einen sehr hilfreichen Grundstein für ein zukünftig stabil laufendes System dar. Um korrelierende Betrachtungsweisen anwenden zu können, reicht es leider nicht, Standardwerte der MIB-II, wie z.B. den Datendurchsatz eines Interfaces, zu beobachten und graphisch darzustellen. Vielmehr wird es unter Umständen notwendig, subnetübergreifend unterschiedliche Werte zu erheben, um dann anhand der daraus resultierenden Ergebnisse Rückschlüsse über Fehlerquellen und Netzwerkphänomene ziehen zu können.
Das Projekt wird mit dem Titel CoVioN versehen. Unterschiedliche Ansprüche an CoVioN werden somit gestellt:

- Es sollte einfach zu bedienen sein
- Neue MIBs sollen leicht importierbar sein
- Die Ergebnisreihen von SNMP-Jobs sollten sich mittels Data-Pipes mit statistischen Operationen variieren und durch graphische Diagramme darstellen lassen.
- Es sollte eine Möglichkeit geboten werden, Standardfälle in so genannte Szenarien abzuspeichern, um bei anderen Nodes gleiche Operationen ausführbar zu machen. So können Standarduntersuchungen auch von instruiertem Personal vorgenommen werden.
- Die Wahl der Persitenzschicht sollte austauschbar und variierbar sein, um eine Integration in andere bestehende Systeme möglicht einfach gestaltbar zu machen
- Es sollten Möglichkeiten geschaffen werden, mühelos Durchsatzgraphiken zu erstellen
- Es sollte einfach sein, einen nicht zufrieden stellend arbeitenden Host dem System hinzuzufügen.
- Über einen MIB-Browser sollte es möglich sein, SNMP-Jobs auf beobachtete Nodes erstellen zu lassen und anzuwenden.
- Eine manuelle Vorgehendweise zur Jobdefinition sollte ebenfalls vorhanden sein.
- SNMP-Jobs sollten sich auch aufgrund von Angabe eines Nodes und eines Szenarios erstellen lassen, ohne den Weg über die manuelle Auswahl von OID’s gehen zu müssen.
- Der Core sollte Schnittstellen bieten, mit dem andere GUI’s (graphical user interfaces) arbeiten können.

Weitere Visionen können dieser Liste noch ergänzen; sie stellt keinen Anspruch auf Vollständigkeit. Im Punkt 8 dieser Arbeit (Zusammenfassung und Ausblick) wird auf das Potenzial dieses Cores noch weiter eingegangen werden.

4.1 Situationsbeschreibung möglicher Zielumfelder

Zielumfeld kann jedes System sein, in dem es sinnvoll sein kann, statistische Werte zu erheben und ausfallkritische Komponenten zu überwachen. Bei VoIP-Netzwerken ist es sinnvoll, noch weitere speziellere Erweiterungen zu kreieren, um z.B. bei einer HG1500 (einem VoIP-Gateway und Gatekeeper der Fa. Siemens) Statistiken über Jitter, Delays und kontinuierlicher Fehlerrate zu bestimmen, um gegebenenfalls bestimmte Teile des Netzwerkes, insbesondere von Routern, besser abstimmen zu können. Da sich die Wartungssituation oft so darstellt, dass Diagnosen getroffen werden müssen, die die Befugnis von Zugriffen auf zwischen liegende Systeme verwehren, lassen sich mittels Vergleiches von Systemparametern der beteiligten Endsysteme trotzdem Aussagen über das dazwischen liegende Netzwerk treffen.
In manchen Branchen ist es üblich Netzinfrastrukturen zu teilen, wenn man sich beispielsweise als ein Unternehmen in einem Bürohaus einmietet. Hier könnten Probleme auftreten, wenn die Hoheit über die Corekomponenten in anderer Hand liegt und sich Fehler nur so weit eingrenzen lassen, als dass man sie als „von Außen“ kommend klassifizieren kann. Gerade im Einsatz von VoIP Applikationen spielt die QoS- Komponente eine tragende Rolle, sobald das lokale Netzwerk verlassen werden soll. Die einzige Variante QoS-Parameter über Providergrenzen hinweg zu bewahren ist es, Service Level Agreements mit den beteiligten Providern vertraglich festzulegen. Stehen im Netzwerk nicht alle Komponenten in eigener Hoheit, so kann man nur noch anhand von Differentialmessungen, die ihre Ansatzpunke bei den Übergabepunkten in fremde Netze haben, versuchen, über Korrelationen die Fehlerquelle außerhalb seines Netzes einzugrenzen, wie folgende Graphik verdeutlichen soll:

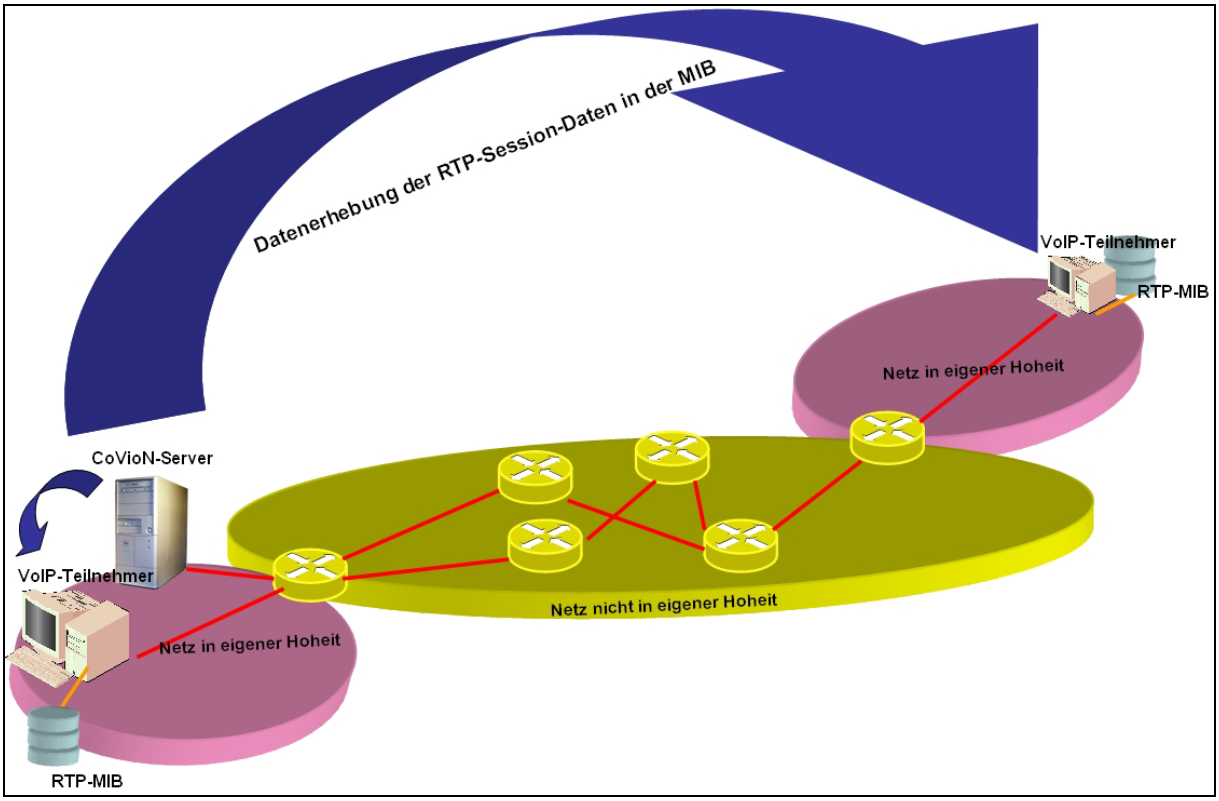

Abbildung 14: Einsatzszenario von CoVioN

Mittels des CoVioN-Servers würde man z.B. eine Datenerhebung mit der Basis der RTP-Mib vornehmen, um Jitter, Delays, Packetloss miteinander graphisch darzustellen und eine Korrelationsbetrachtung aufgrund der Graphiken vorzunehmen.

4.2 Ein Lösungsansatz mit Korrelationsbildung

Die ersten Vorstellungen von Oberflächen, die an Netzwerkmanagement erinnerten, entwickelten sich zunächst in die Richtung von netzwerkzustandsbehafteten Ampelsignalisierungen. Jede verwaltete Netzwerknode hält unterschiedliche Watermarks für qualitätsbehaftete Merkmale, wie z.B. einen Ping, bereit. Die RTT konnte hierbei unterschiedlichen Ampelphasen entsprechen, die kontinuierlich von einem aktualisierenden Thread gemessen und vorgemerkt wurden.
Denkbar wäre es auch andere Thresholds auszusuchen und anzuwenden. Bei einem Router mit vielen Interfaces wäre es beispielsweise interessant, Watermarks für unterschiedliche Queuelängen zu setzen. In der MIB-II wurde hier der Wert „1.3.6.1.2.1.2.2.1.21.x“ (x=

Interfacenummer) angeboten, der eigentlich von jedem Router-Agenten implementiert sein sollte. Man könnte somit je nach Device-Klasse andere, aussagekräftige Werte nehmen.

Als Standardimplementierung wurde in diesem Projekt ein Ping-Server implementiert, der als eigener Thread periodisch alle angemeldeten Nodes pingt und asynchron die letzen Antwortzeiten der jeweiligen Nodes bei Bedarf liefert. Auf diese Weise lässt sich in der View die Darstellung mittels Ampeln oder anderer Visualisierung verwirklichen.
Graphiken lassen sich besser für das menschliche Auge und Geist aufnehmen und Zusammenhänge mit einem bloßen erkennen, ganz im Gegensatz zur tabellarischen Darstellungsweise.
Der Ansatz von CoVioN sieht vor, dass Jobs definiert werden, die auf der Basis von SNMP-Anfragen in regelmäßigen Abständen Daten sammeln und persistieren. Die aus den Jobs hervorgegangenen Daten werden aufbereitet und mit anderen Jobergebnissen gemeinsam in Graphiken dargestellt.
Die Frage der Anwendbarkeit stellt sich von Gerätschaft und Infrastruktur jedes mal neu. Die ganze Architektur von CoVioN steht und fällt mit dem Umfang der implementierten MIB's, die das Equipment in Form von SNMP- Agenten bereitstellt. Die Integration von neuen Mibs in CoVioN stellt sich als einfach dar. Ein definiertes Verzeichnis birgt alle MIB-Dateien, in das man auch weitere integrieren kann.
Bei einer Implementierung von RFC 2959 (RTP-Mib, [24]) würde man beispielsweise die Möglichkeit erlangen, RTP-Sessions zu verfolgen, Jitter, Delays u.ä. kontinuierlich zu erfassen und Testszenarien mit der Zeit zu standardisieren, um häufig auftretende Fehler schnell einzugrenzen und zu lokalisieren.

„[…]RTP Monitors may use the RTP MIB to collect RTP session and stream statistical data; […] An RTP Monitor may use the RTP MIB to collect data to permit a network manager to detect and diagnose faults in RTP sessions or to permit a network manger to configure its operation.[…]"
(M. Baughter, B. Strahm, RFC 2959, http://www.ietf.org/rfc/rfc2959.txt [24])

Natürlich haben Hersteller auch spezifische MIBs zu ihren Systemen verfasst und entsprechende Agents auf den Komponenten integriert. Das VoIP-Gateway und gleichzeitig Gatekeeper mit der Bezeichnung HG1500 hat ihre eigene MIB als Agenten implementiert, die sich als Mibtree unter dem Knoten „.1.3.6.1.4.1.231" einordnet. Sie besteht aus mehreren Tabellen, die

Trapinformationen, Statistiken über VoIP, geführte VoIP-Anrufe, Systemstati usw. beinhaltet. Da die HG1500 die Brücke von IP zu PSTN darstellt, beinhaltet sie insgesamt 5 Interfaces, die jeweils ihre eigene Spezialisierung darstellen. Unter anderem werden von Agent Daten über das ISDN-Interface, QoS-Parameter und auch allgemeine Statistiken von Voice-Gesprächen aufgezeichnet. Folgendes Schaubild soll einen Überblick vermitteln.

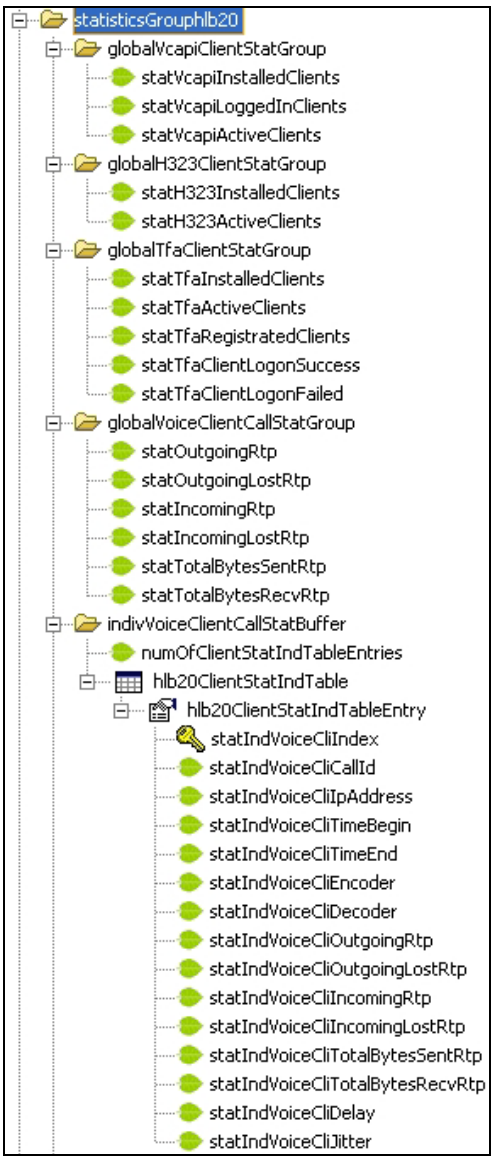

Abbildung 15: Ausschnitt aus der HG1500-MIB

Der Agent hält standardmäßig eine Statistik der letzen 100 geführten Gespräche vor und ermittelt zu den Gesprächen die Teilnehmer-IP, Beginn und Ende eines Gespräches, wie viele Pakete gesendet wurden usw. Diese Zusammenstellung von Daten könnte alleine eine auf SNMP basierende Applikation motivieren, die sich alleine auf die Nutzung der HG1500-MIB konzentriert.

5. Funktionalitäten und Grundkomponenten für CoVioN

Die Anwendungsarchitektur von CoVioN besteht aus 3 Schichten, einer View, die abgelöst vom Core implementiert werden muss, der mittleren Schicht dem Core und einer leicht austauschbaren Persistenz als dritte Schicht. Das folgende Schaubild in Abbildung 16 illustriert zunächst den Kern mit seinen Hauptkomponenten in Verbindung mit der Persistenzschicht.

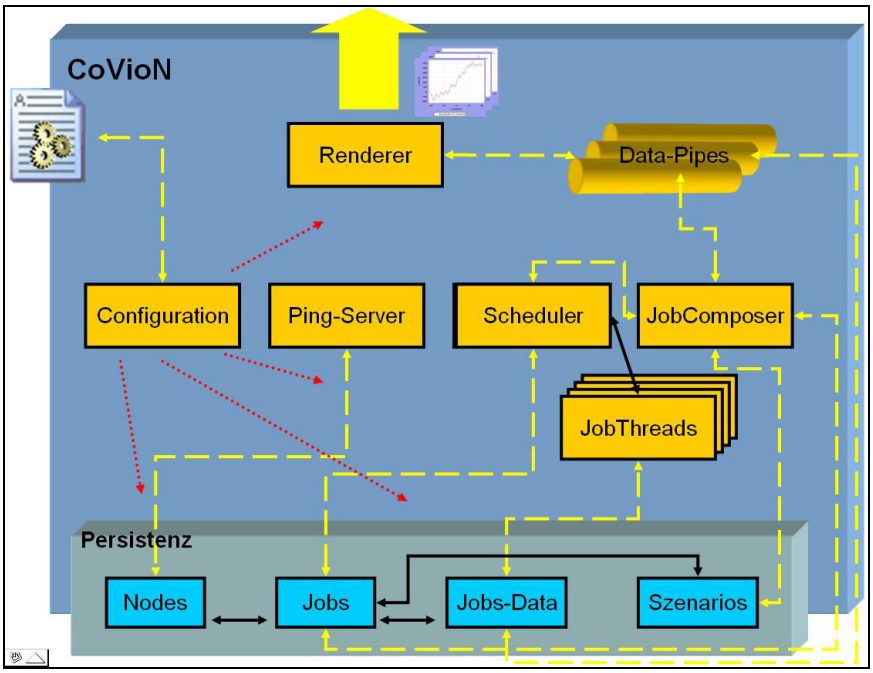

Abbildung 16: Darstellung des CoVioN- Konzeptes

Das Konzept sieht eine austauschbare oder sogar gemischte Persitenzschicht vor. Wie schon zuvor in Punkt 4 angedeutet soll die Möglichkeit geschaffen werden, die Komponenten der Persistenz konfigurierbar und damit austauschbar zu machen, um zum einen eine Integration des Systemcores in andere Systeme möglichst einfach gestaltbar zu machen und die Wahl des Persistenzbackends nicht einem möglichen Kunden abzunehmen. Im Gegenteil, in manchen Firmen sind bestimmte Persistenzschichten bereits etabliert. Warum sollte man ein neues Datenbank Management System kaufen, wenn man schon ein gut funktionierendes besitzt. Hierzu wird eine Einheit benötigt, die die konfigurierten Daten allen Komponenten bei Bedarf zur Verfügung stellt; durch „Configuration“ dargestellt. Ein Ping-Server beobachtet die Erreichbarkeit aller konfigurierten Nodes und interagiert mit der View-Komponente, sobald Nodes ausfallen oder angegebene RTT nicht eingehalten werden. Dies kann z.B. mittels einer Ampelsignalisierung geschehen. Über den Scheduler können neue Jobs definiert und erschaffen werden. Dieser

veranlasst, dass Theads gestartet werden, die die Sammlung der Daten vornehmen. Der Scheduler interagiert mit den Threads, um die Daten der Jobs und ihre Zustände zu synchronisieren. Dazu wird später mehr im Punkt 5.7.4 eingegangen, da dort die Zustandsdiagramme erklärt und illustriert sind. Die Job-Threads erzeugen die Job-Daten, die wiederum aufgegriffen werden, um direkt über eine Data-Pipe zu einem Renderer zu gelangen, der aus den Daten Diagramme für die darüber liegende View-Komponente erzeugt. Mittels Composer sollen Szenarien zusammenstellbar sein, die dazu genutzt werden können, um im Laufe von Wartungsarbeiten standardisierte Untersuchungen möglichst einfach auf andere Nodes anwenden zu können. Zudem soll die Möglichkeit geboten werden Rohdaten mit statistischen Methoden zu bearbeiten und diese dann in Kombination mit den Rohdaten zu rendern.

5.1 Userinterface

Die Eingangssicht von CoVioN stellt eine Übersicht zur Verfügung, in der er alle eingegangenen Traps, sowie alle verwalteten Hosts aufgeführt sind.
In der ersten Version haben wir die Möglichkeiten, unterschiedliche Interaktionen auszuführen:

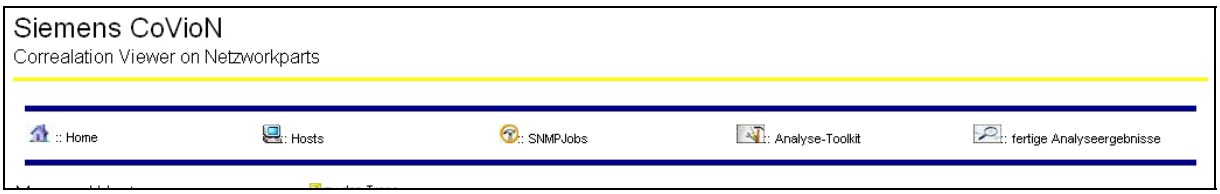

Abbildung 17: Funktionsleiste des Anwender-Interfaces

Es besteht die Möglichkeit unterschiedliche Verwaltungen aufzusuchen. Host-Verwaltung, SNMP-Job-Verwaltung, Analyse-Toolkit und fertige Analysen.

Host-Verwaltung

In der Hostverwaltung besteht die Möglichkeit typische add, edit und delete – Operationen auszuführen. Darüber hinaus kann man an einem Host einen Standard-Test vorzunehmen. Hierbei wird zunächst Versucht den Host an zu pingen und danach über die angegebenen Angaben eine SNMP-Anfrage auf die Systemumschreibung mit dem Key „.1.3.6.1.2.1.1.1.0“ vorzunehmen – ein Wert, der jedes System, der die Standard- MIB-II implementiert zulässt. Besteht der Host beide Tests, wird er als „pingable“ und „manageable“ markiert.

SNMP-Job-Verwaltung

Hier werden SNMP-Jobs entweder manuell oder aber mit Hilfe des Mibbrowsers erstellt. Bevor ein Job initialisiert wird, kann noch eine Testanfrage vorweggeschickt werden, um den weiteren Jobverlauf vorher zu erproben. Derzeit ist es nur möglich, statische Mib-Einträge zu definieren. In weiteren Ausbaustufen soll auch die Möglichkeit eröffnet werden, über alle Reihen iterierend Spalteneinträge zu ermitteln und Mittelwerte über diese zu bilden. Mit Bezug auf Abbildung 15 wäre es denkbar, über alle Gespräche eine ständige Mittelwertbildung über alle Jitterwerte der Gespräche zu erheben, um ein Gesamteindruck des Systemzustands bereitzustellen.

Analyse-Toolkit

Im Analyse-Toolkit stellt man die nachfolgenden Testergebnisse zusammen und kann sie gegebenenfalls noch mit Auswertungskomponenten, genannt Data-Pipes bearbeiten. Data-Pipes sind hierbei Daten verändernde Einheiten, die z.B. aus einer gegebenen Datenreihe eine Datenreihe mit kontinuierlicher Mittelwertbildung erschafft. Das Konzept sieht an dieser Stelle eine einfache Erweiterbarkeit vor, indem für jedes Anwendungsgebiet von Daten verändernden Operationen eigene Pipes integriert werden können. Das zu implementierende Interface regelt, dass unterschiedlicher Pipes beliebig miteinander kombinierbar sind.

Fertige Analyseergebnisse

Hier werden alle Ergebnisplots, bzw. Ergebnisgraphiken dargestellt. Sie gehen aus den Szenarien oder Rohdaten des Analyse-Toolkits hervor.

5.2 Trapmanager

Der Trapmanager hat die Aufgabe, alle PDU-Type 4 Nachrichten, die an ihn gesendet werden, zu sammeln und Methoden zu definieren, die der View die Möglichkeit bietet, Traps mit einem Bearbeitungsstatus, wie z.B. „neu“, „in Bearbeitung“ und „abgeschlossen“, zu versehen. Ein Trap würde also an das System mit folgenden Paket-Informationen gesendet werden.

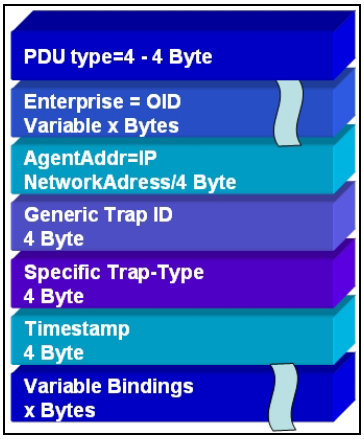

Abbildung 18: SNMP Paketaufbau Trap v1

Der Paketaufbau von Traps in SNMPv1 beinhaltet ein Enterprise Feld, welches standardmäßig den Wert der Systembeschreibung der MIB-II beinhaltet, die den Trap verursacht hat, sowie die IP-Adresse des Agents, der den Trap gesendet hat. Die Felder Generic Trap und Specific Trap tragen eigentlich dieselbe Bedeutung in sich, jedoch stammt die Semantik der Generic Trap ID aus dem RFC 1215 [7], welches eine Darlegung von zu Implementierenden Trap-Arten ist. Das Specific Trap-Type hingegen ist implementierungsabhängig und je nach Hersteller anders belegt, da hier die Traps der eigenen private MIB's mit einfließen. Der Timestamp wird auch seitens des Agenten gesetzt. Weitere Variable-Bindings beschreiben den Ursprung der problematischen MIB-Instanz; dies kann z.B. ein Interface sein, das seinen Zustand geändert hat.
Von einer Persistierung soll abgesehen werden, da dies Daten sind, die eher kurzlebig sein sollten. Traps, die vor Jahren gesendet worden sind, sind nicht von Interesse. Trotzdem wäre es möglich, auch eine Persistenz für diese Traps zu bewerkstelligen.

5.3 SNMP Job- Verwaltung

Die SNMP-Job-Verwaltung zeigt zunächst eine Übersicht über alle geführten Jobs und Informationen über selbige. So z.B. die OID die abgefragt wird, Start- und Endzeitpunkt und einen State, der Aufschluss darüber gibt, in welchen Zustand sich der Job befindet.

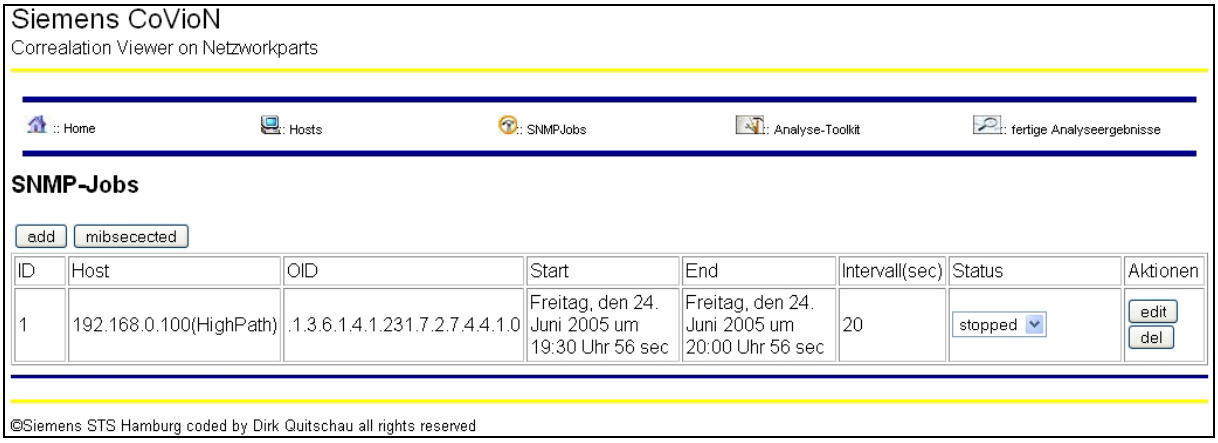

Abbildung 19: Darstellung Übersicht Job-Verwaltung

Diese Sicht bietet die Möglichkeit, neue Jobs zu generieren, alte Jobs zu löschen (dies beinhaltet allerdings, dass auch die Ergebnisse, die aus diesem Job hervorgingen mit gelöscht werden) oder zu editieren (nur möglich solange der Job noch nicht gestartet wurde). Bei Operationen an Daten muss die Datenkonsistenz, in Bezug auf das Löschen von Daten genauso beachtet werden, wie auch ein Locking-Mechanismus, dessen Steuerung anhand der Job-Zustände festgemacht ist.
Um neue Jobs zu generieren bestehen derzeit zwei Möglichkeiten von geplanten drei offen:
Die manuelle Art über den Button „add“:

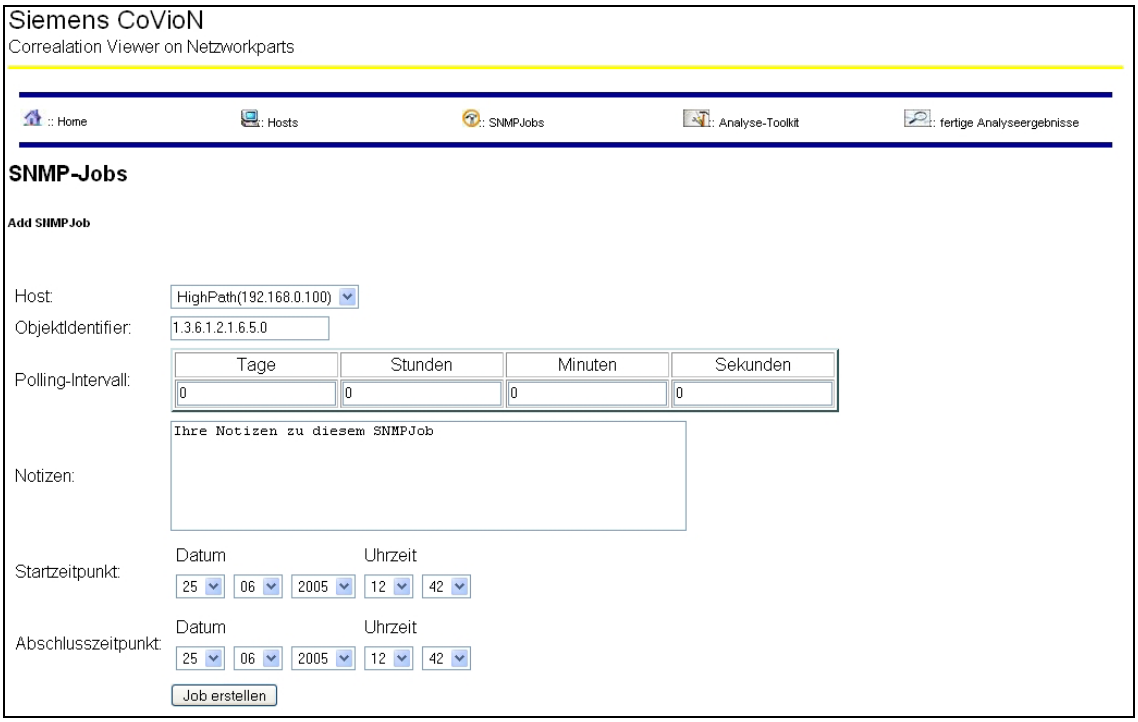

Abbildung 20: Manuelle Jobdefinition

Hier kann man unter dem Punkt Host einen Node auswählen, einen ObjektIdentifier editieren und ein Polling-Intervall definieren, in dem der SnmpJobThread die angegebene OID-Instanz abfragt. Ein Feld für Notizen wurde auch vorgesehen, sowie Start- und Endzeitpunkt der Überwachung.

Sowie die „mibselected“ Art, wo die Auswahl der OID und des zu überwachenden Hostst komfortabel über einen Mib-Browser ausgewählt wird:

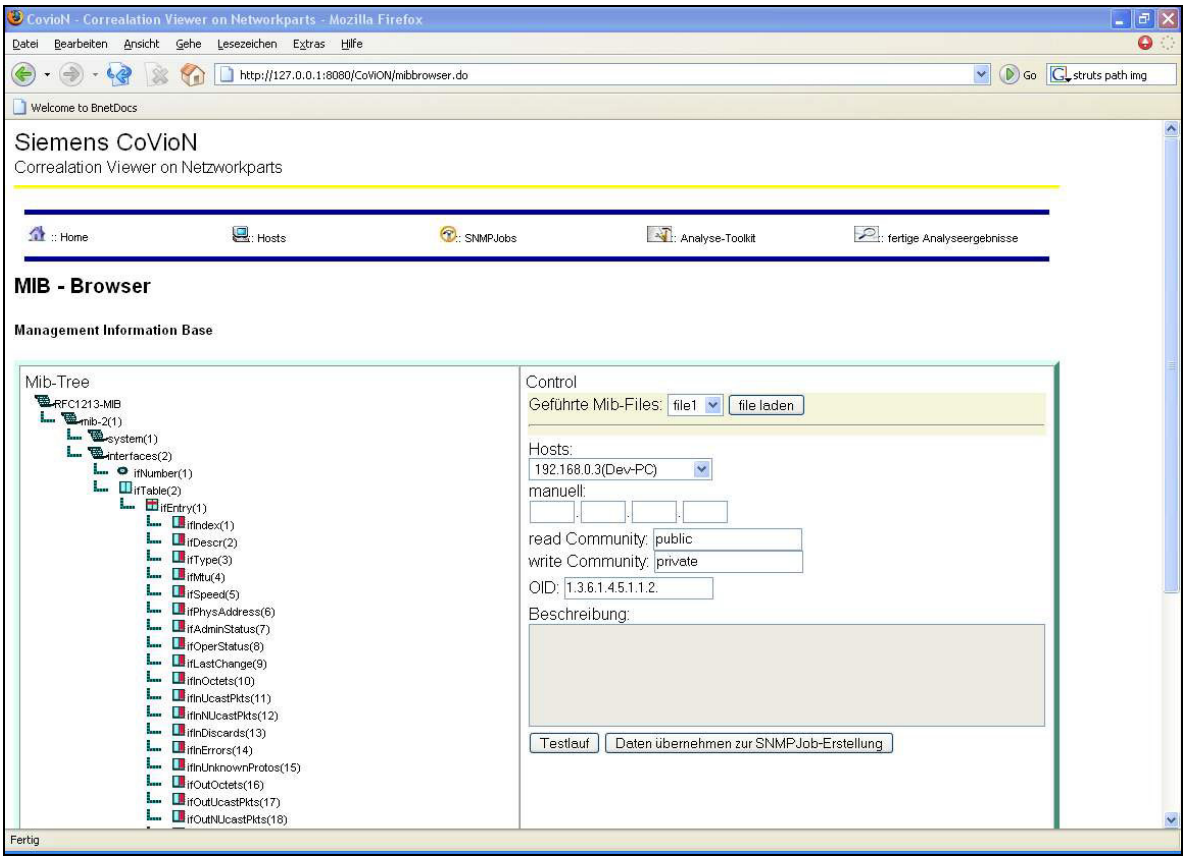

Abbildung 21: Jobdefinition über den Mibbrowser

Hier wird eine MIB geladen und kann gebrowsed werden. Die hier angewandte Technik baut auf JSP- und Servletstechniken auf und basiert auf erweiterten Klassen, die vom freien MIB-Parser „Mibble“ (http://www.mibble.org [3]) abstammen. Tiefergehende Erörterungen zur Funktionalität werden in Punkt 6 – Implementation vorgenommen werden.

Die dritte, noch nicht implementierte Art, Jobs zu definieren besteht in dem Ansatz Szenarien an den Scheduler zu übergeben, der daraufhin alle relevanten Jobs automatisch generiert. Die hierzu notwendigen Daten müssten aus Job-Clonen in Verbindung mit Data-Pipes bestehen, die

in einem Verarbeitungsplan hintereinander ausgeführt werden sollen. Zu diesem Zweck ist eine Datenstruktur geschaffen worden, die eine semantische Aussagekraft besitzt, um bei einer Übergabe an den Scheduler, diesen dazu zu bewegen, eine Abfolge von Operationen aufgrund des erhaltenen Objektes auszuführen, um die beinhaltenden SNMP-Jobs zu generieren. Darüber hinaus ist es notwendig den Scheduler mittels Rückmeldungen einzelner SNMPJobThreads eine Steuerung der Szenario-Zustände zu implementieren. Ein Szenario hat natürlich erst seinen Endstatus erreicht, wenn alle beteiligten SNMPJobs den Endzustand erreicht haben. Erst dann sind alle relevanten Datenerhebungen vollbracht und die einzelnen Pipes können auf die Daten angewendet werden. Ein möglicher Implementierungsansatz kann sein, ein Szenario als eine Art FIFO-Queue zu implementieren. Die Queue wird mit Inhalt über die Methode mit der Signatur

```
public boolean addConfig(SzenConfig conf);
```

gefüllt. Hierbei spielt der Rückgabewert, der Boolean eine gesonderte Rolle. Die Klasse SzenConf beinhaltet nur eine Pipe, mit der Daten verarbeitet werden sollen. Die Klasse SzenConf wird von zwei weiteren Klassen abgeleitet; der Klasse JobConfig, die einen SNMPJob-Clone enthält, und StackConfig, die dem Szenario quasi mitteilt, dass die abgeleitete Pipe auf eine Ergebnisreihe angewendet werden soll, welche von einem vorangegangenen JobConfig-Objekt resultiert, sie stellt nur eine Markerklasse dar. An dieser Stelle wird auch bei weiteren Überlegungen klar, warum man den Boolean Rückgabewert benötigt. Dieser signalisiert, ob das SzenConfig-Objekt überhaupt Sinn macht. Natürlich ließe sich keineStackConfig auf eine Ergebnisreihe anwenden, wenn vorher nicht mindestens ein JobConfig-Objekt geadded wurde. StackConfig fungiert somit nur als Markerobjekt für die später verarbeitende Klasse.

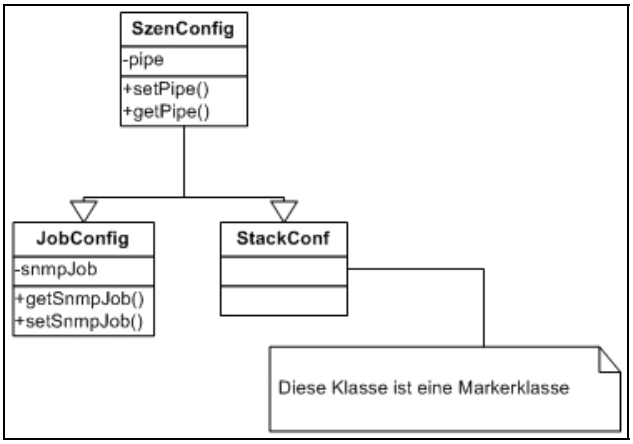

Abbildung 22: Klassendiagram der SzenConfig-Entitäten für ein Szenario

5.4 Netzwerkknoten - Verwaltung

Wie bei der Jobverwaltung beginnt auch die Nodeverwaltung mit einer Übersicht über alle geführten Nodes.

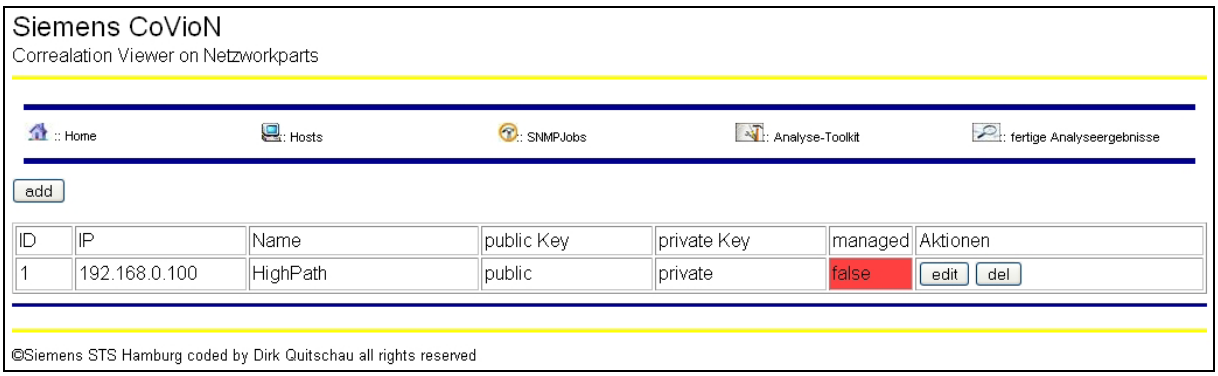

Abbildung 23: Node-Übersicht

Hier lassen sich Nodes hinzufügen editieren und löschen. Beim Hinzufügen erhält man folgende Maske:

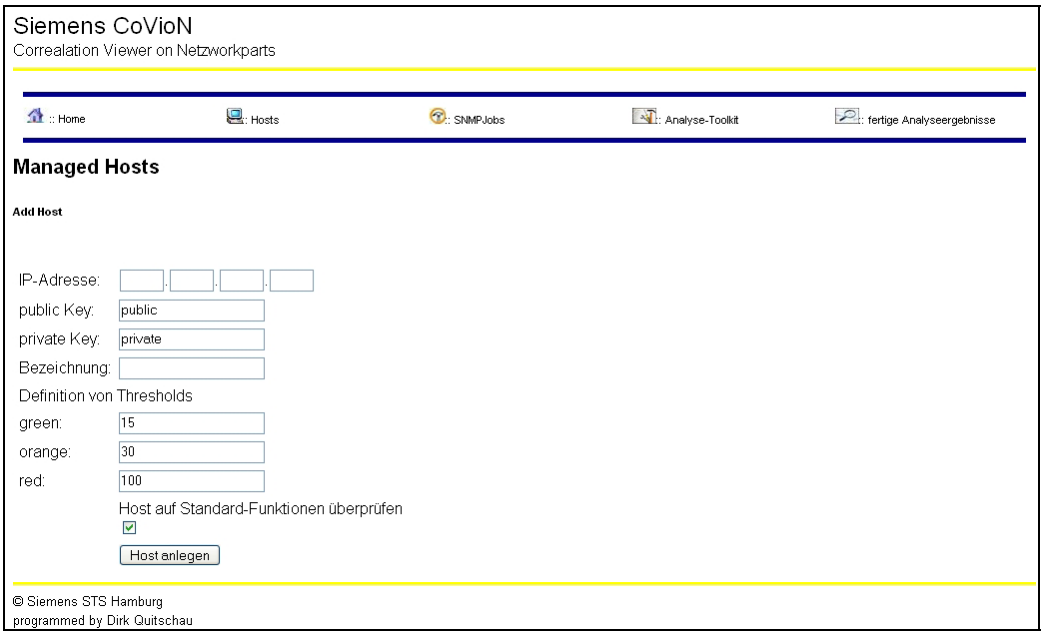

Abbildung 24: Eingabemaske, hinzufügen eines Hosts

Die Felder sind weitestgehend selbsterklärend, bis auf die Felder green, orange und red. Diese Felder sollen Thresholds darstellen, die vom Pingserver (siehe ggf. Abb. 16 bei Punkt 5) geprüft werden und entsprechend einer Ampel dargestellt werden können. Diese Felder sind derzeit

schon vorgefertigt, werden jedoch noch nicht vom Prototyp genutzt. Die Checkbox „Host auf Standard-Funktionen überprüfen“ soll zukünftig dazu dienlich sein, der Hinzufügung des Host eine Prüfung voraus zu senden, um beispielsweise die Thresholds schon vor der Erstellung zu justieren. Im Falle von einem Threshold des Pings könnte man die Antwortzeiten als Anhaltspunkt nutzen, um die Threaholds festzulegen. Ein Node ein eigener Broadcastdomäne wird sicherlich andere Thresholds haben, als ein Node, der in weiter entfernten Netzwerken angesiedelt ist.

Beispiel:
Bei einem Test eines Hosts im eigenen Netzwerkes erhält man 1 ms, dann würde man die Thresholds beispielsweise mit grün=2ms; orange=200ms und red=1000ms festlegen, um sinngemäß die Ampelphasen für diesen Host festzulegen. Bei einem Host, der sich nicht im eigenen Netzwerk befindet und eine Antwortzeit von 80ms beim Test vorweist, würde man green=100ms, orange=300ms und red=2000ms festlegen.

Ein Aufruf der Eingangseite könnte z.B. daran gekoppelt sein, dass die zuständige ActionKlasse, bevor sie die Seite zur View weiter leitet, sich zunächst über die letzen Antwortzeiten aller Nodes beim Pingserver erkundigt, die Werte für jede Ampel vorbelegt und dann weiter sendet. Somit würden die Ampelphasen bei jedem erneuten Request der Seite aktualisiert werden.

5.5 MIB-Browser

Ein MIB-Browser hat die Fähigkeit ASN.1 codierte Dateien einzulesen und die darin befindlichen Objekte ein einer View in einer Baumstruktur zur Verfügung zu stellen. Viele MIB-Browser unterstützen zusätzlich ad-hoc Anfragen an entfernte Systeme.
Wie bei der Beschreibung der Joberstellung schon angedeutet, verfolgt das Implementierungskonzept von CoVioN einen webbasierten Ansatz. Hierbei macht sich CoVioN das Session-Objekt zu Nutze, um die Baumstruktur der MIB nach jedem aktivieren einer Entität die Visibility des Baums neu zu berechnen, und zu rendern.
Bei einer Implementierung einer View mit einem anderen Graphik-Framework sollte man sich den Tree-Implementierungen dieser bemächtigen und nutzen. Ein Ansatz kann hierbei der SWING-Ansatz JTree sein, der auch von Mibble selbst bei seiner Implementierung seines Browsers nutzt. Diesbezüglich ist es sicherlich lohnenswert die Dokumentation der SWING-Implementation von Mibble in seine Betrachtungen mit einzubeziehen.

Wie man schon an der Eingangsgraphik der Einleitung des 5. Kapitels Abb. 16 vielleicht schon entdeckt hat, ist der MIB-Browser nicht Teil des Cores. Er gehört zu den Annehmlichkeiten, die eine View bereitstellen sollte, um nicht die parallele Nutzung eines separaten MIB-Browser bei der Joberstellung voraussetzen zu müssen.

5.6 Weiterverarbeitung der SNMP-Rohdaten

Nach dem Akkumulieren von Daten durch sämtliche SNMP-Jobs müssen die Daten weiter verarbeitet und für die Graphikerstellung noch in die richtige Form aufbereitet werden.
Als Renderer wird auf das Projekt Cewolf [2] zurückgegriffen, welches bereits eine Weiterentwicklung eines anderen GPL Projektes darstellt. Cewolf ist eine Taglibrary, die einen Renderer in der Form eines Servlets bereitstellt. Cewolf baut hierbei auf JFreeChart auf. Hierbei werden bestimmte Datenformate für eine Nutzung des Renderers vorausgesetzt. Es wird vorgesehen das Interface DatasetProducer zu implementieren, welches wiederum voraussetzt, dass eine Methode produceDataset() implementiert wird:

```
public Object produceDataset(Map params) throws DatasetProduceException {}
```

In der Methode wird erwartet, dass ein Dataset vom Typ org.jfree.data.general.Dataset zurückgegeben wird. Da wir in unseren Anwendungsfällen eigentlich nur mit zeitkontinuierlichen Werten zu tun haben, hat die Klasse de.quit.covion.data.pipes.CovionDatasetProcessor die Methode so implementiert, das ein Array von SNMP-Job-Rohdaten in ein org.jfree.data.TimeSeriesCollection transferiert und im Anschluss an den Renderer von Cewolf sendet wird.

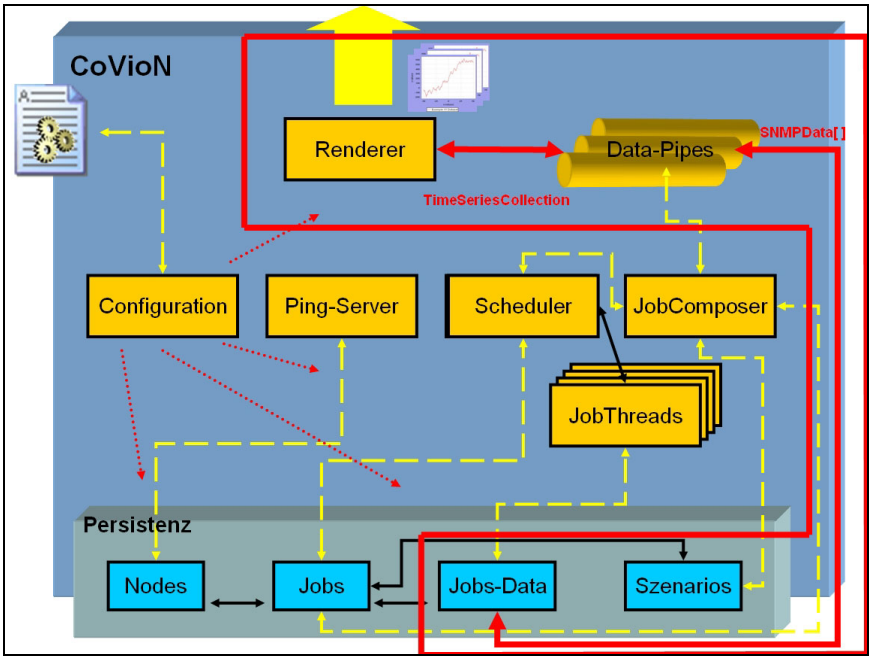

Abbildung 25: Datenwandlung für Cewolf-Renderer

Auf die konkrete Umsetzung wird in Punkt 6, Implementation eingegangen werden.

5.7 Architektur des Cores

5.7.1 Schichtenmodell

CoVioN's Schichtenmodell teilt es sich in drei Schichten auf und versucht dem MVC (model view control)– Model, welches eine der weit verbreiteten und bekanntesten Softwaregrundideen ist, gerecht zu werden.

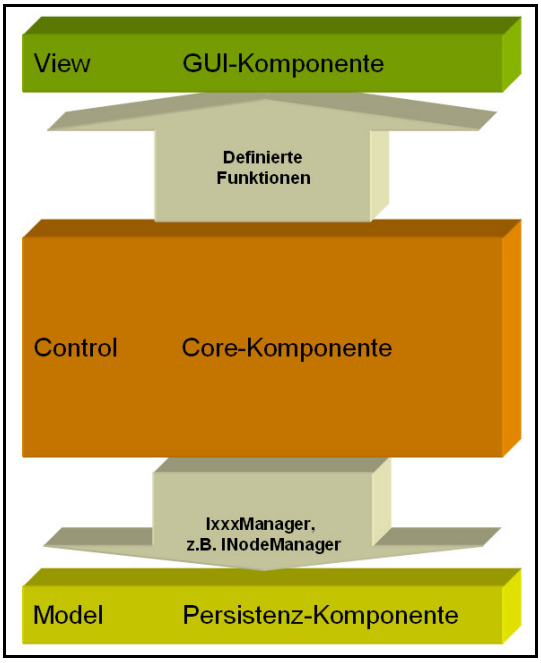

Abbildung 26: Darstellung der Schichtenarchitektur mit Coreschnittstellen

Das Multitier-Modell setzt voraus, dass schon vorhandene Datensenken genutzt werden können. Darüber hinaus weist es eine hohe Flexibilität in der Ankopplung unterschiedlicher GUI-Komponenten vor. An der Kontrollereinheit eines solchen Systems ist der Ort, an dem all diese Anforderungen zusammengeführt und umgesetzt werden müssen.

5.7.2 Ansteuerung des Kerns

Schon zu Beginn des Projektes gab es den Wunsch, die Persistenzschicht austauschbar machen zu können, um eine Möglichkeit zu schaffen, sich möglichst variabel in schon vorhandene Softwarearchitekturen integrieren zu können. In Systemhäusern ist es üblich, dass schon DBMS (Datenbank Management Systeme) existieren und somit nur Software angenommen wird, die eine Portabilität der Persistenzschicht vorweisen können.
Um die Portabilität zu erreichen fiel die Entscheidung darauf, eine Konfiguration mittels XML vorzunehmen. Definiert sei eine XML-Datei mit dem Namen ***covionconf.xml,*** welche sich im Unterverzeichnis ***conf*** der Applikation befindet.
Die DTD für die Konfigurationsdatei lautet:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE covion [
<!ELEMENT covion (conf+)>
<!ELEMENT conf (param,value)>
<!ELEMENT param (#PCDATA)>
```

```
<!ELEMENT value (#PCDATA)>
]>
```

Daraus geht die Bildung von bestimmter Tag-Reihenfolgen hervor:

```
<covion>
        <conf>
                <param>Key-String</param>
                <value>Value-String</value>
        </conf>
</covion>
```

Das Konzept der Konfigurierbarkeit sieht weiterhin vor, dass im Startup-Code des Programms die Klasse StartupParams instanziiert wird, die wiederum die Verarbeitung des XML-Files vornimmt und Key-Value-Paare mit der Methode getApplicationProperty() für alle Komponenten verteilt. Entwickler können bei der weiteren Entwicklung von CoVioN ihre eigenen Key-Value-Paare in die Konfigurationsdatei mit einbringen und über die Klasse StartupParams erreichen.
In Umgebungen mit vielen Threads, die konkurrierend auf Ressourcen, wie z.B. der Datenbasis zugreifen, ist es sinnvoll Verwaltungseinheiten als Singelton zu implementieren, um Datensynchronisation nicht unnötig komplex zu gestallten. In Java gibt es die Möglichkeit sich eine Instanz einer Klasse über einen Class-Aufruf zu erhalten. Dieser Mechanismus wird beispielsweise oft bei der Instanziierung von JDBC-Treibern genutzt.

```
Class.forName(´<vollqualifizierter Name>´).newInstance();
```

Das nun folgende Bild illustriert die Vorgehensweise, wie CoVioN seine Persistenzschicht konfigurierbar und somit austauschbar macht. Dabei wurde ein Interface für jede Managerklasse definiert und darüber hinaus auch ein Beaninterface. Beide Interfaces müssen bei einem Austausch der Persistenz implementiert und zuletzt Konfiguriert werden.

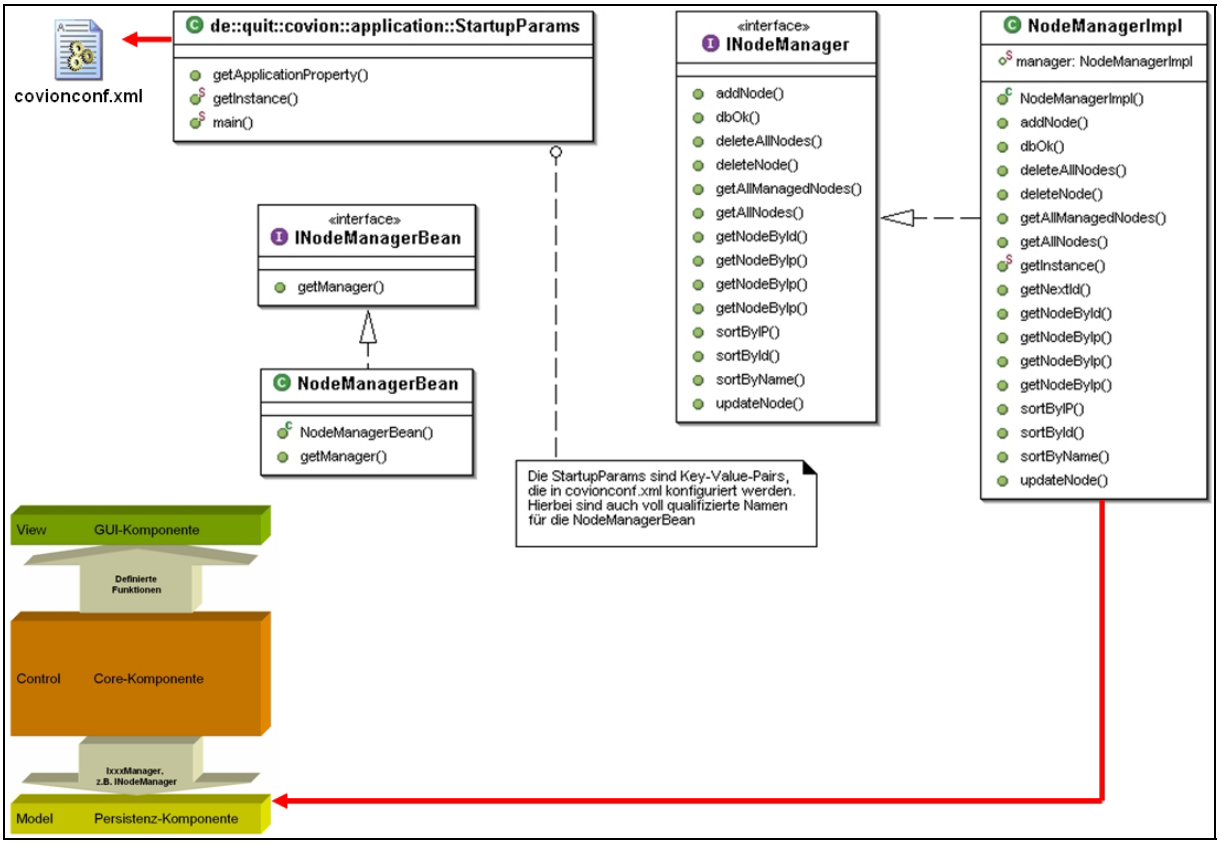

Abbildung 27: Ladevorgang für die konfigurierten Manager der Persistenzschicht am Beispiel INodeManager

Die Ansteuerung des Kerns geschieht zum einen über die Persister-Klasse, die sinngemäß alle Managerklassen zusammenfasst und nach außen zur Verfügung stellt. Soll ein Job der Jobverarbeitung zugefügt werden, so wird dies über die Methode addNewSnmpJob() des Schedulers gesteuert. Alle damit verbundenen Threads werden instanziiert und verarbeitet.

5.7.3 Klassendiagramme

Die Persistenzschicht wird von CoVioN mittels unterschiedlicher Manager-Interfaces beschrieben, die allerdings in ihrer grundsätzlichen Konzeption nicht sehr unterschiedlich sind. Alle zu persistierenden Daten werden durch ihre jeweiligen Manager in die Datenbasis integriert. Sämtliche Interaktionen von Daten, wie das Hinzufügen, Ändern und Löschen derselbigen, wird über den Manager gesteuert. Außerdem sieht jedes Interface vor, dass ihre Entitäten auf unterschiedliche Art und Weise sortiert werden können. Diese Arbeit wird den View-Komponenten abgenommen.

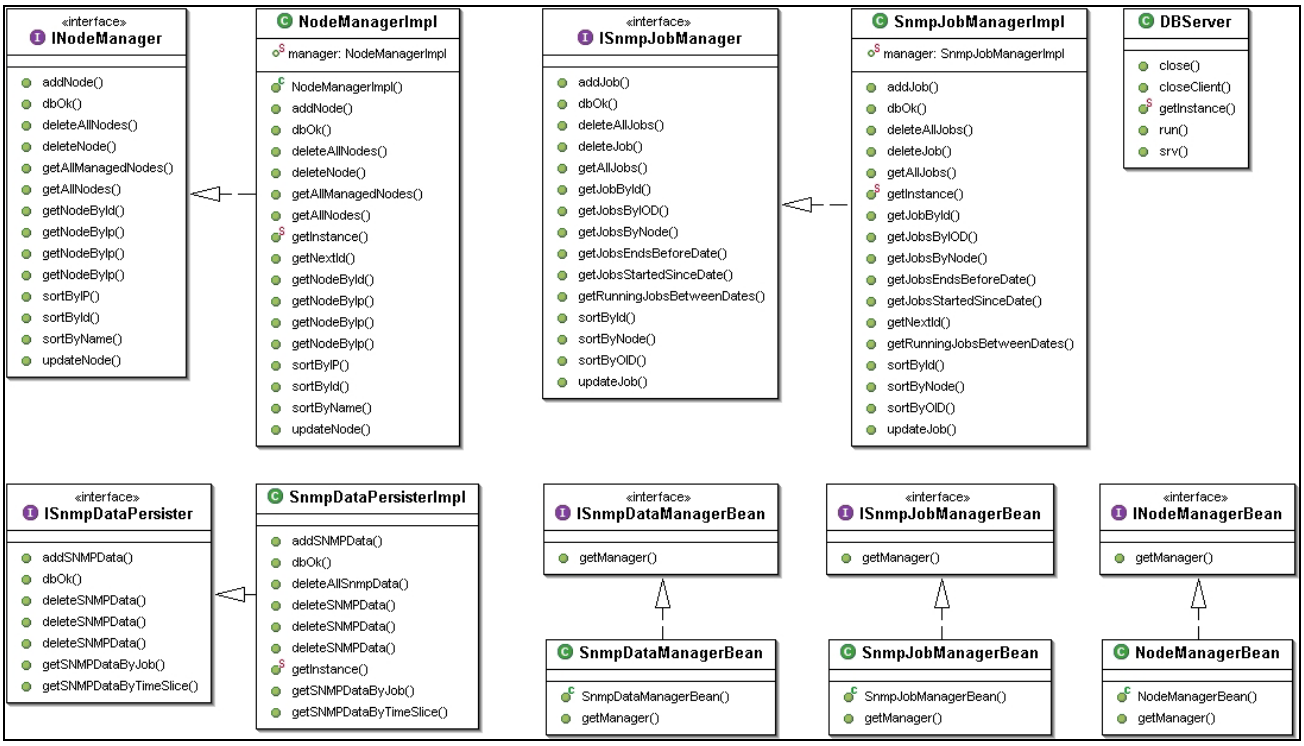

Abbildung 28: Klassendiagramm der Persistenzschicht

Der Scheduler und die daraus folgenden Threads haben folgende Klassenstruktur.

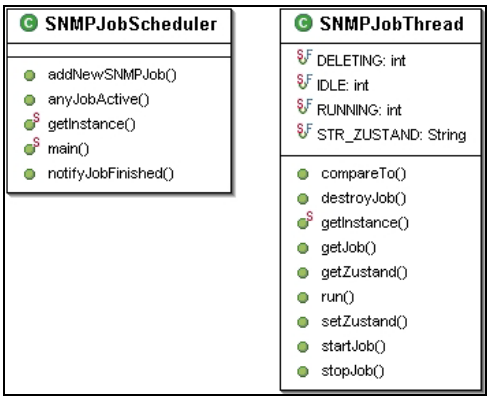

Abbildung 29: Klassendiagramm des Schedulers und der Job-Threads

Die Interaktion beider Klassen wird in den Sequenzdiagrammen und Zustandsautomaten ausführlich erklärt.

Die Konfiguration wird über ein xml-File und deren Key-Value- Paare definiert. Allen Komponenten werden sie mit der Kasse StartupParams zur Verfügung gestellt.

Abbildung 30: Konfigurationsklasse

Alle Komponenten bekommen ihre benötigten Initialisierungsparameter über den Aufruf:

```
String value = StartupParams.getInstance().getApplicationProperty(<key>);
```

5.7.4 Sequenzdiagramme und Zustandsautomaten

Ein Snmp-Job-Thread funktioniert nach folgendem Zustandsdiagram:

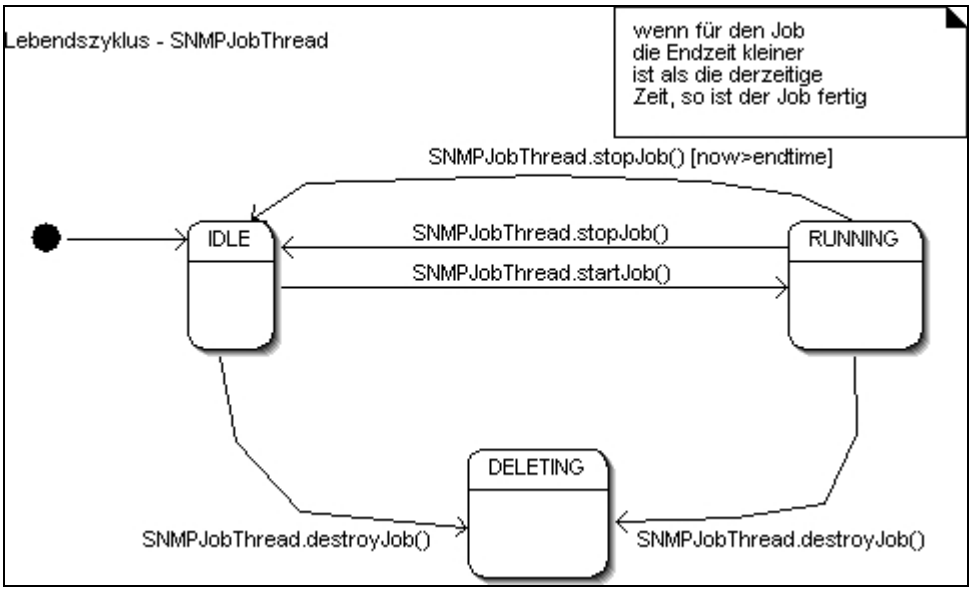

Abbildung 31: Zustandsautomat für SNMPJobThread

Wird der Thread gestartet und die Run()-Methode wird ausgeführt, befindet er sich im Zustand *Idle*. Das bedeutet, dass seine eigentliche Arbeit, das Sammeln von SNMP-Informationen, noch nicht begonnen wurde.
Wenn der Thread vom Scheduler gestartet wird (Aufruf: startJob()), wechselt er in den Zustand *Running*. In *Running* vollzieht er seine Arbeiten, kann aber dabei mit einem stopJob()-Aufruf vom

Scheduler wieder in den Zustand *Idle* zurückgesetzt werden. Dieser Aufruf könnte von einer Aktion in der GUI ausgelöst werden, in *Idle* unterbricht der Thread seine Aufgabe. In welcher Weise die Arbeit verrichtet wird, wird im Implementationsteil dieser Arbeit unter dem Punkt 6.2 näher erörtert werden.
Mit jedem SNMP-Job wird auch ein Zeitintervall vorgegeben, in dem die Daten vom Thread zu erheben sind, und in welchen Abständen dies geschehen soll. Ist der Endzeitpunkt für einen Job überschritten, veranlasst der Thread einen Zustandswechsel zurück in den Zustand *Idle* und lässt dem Scheduler eine Benachritigung zukommen, dass er den Job beendet hat. Der Scheduler wird daraufhin die destroyJob()-Methode vom fertigen Thread aufrufen, die eine selbst initialisierte Beendigung des Threads zur Folge hat.

Was nun eine Erstellung eines Jobs zur Folge hat lässt sich gut in folgendem Sequenzdiagramm nachvollziehen.

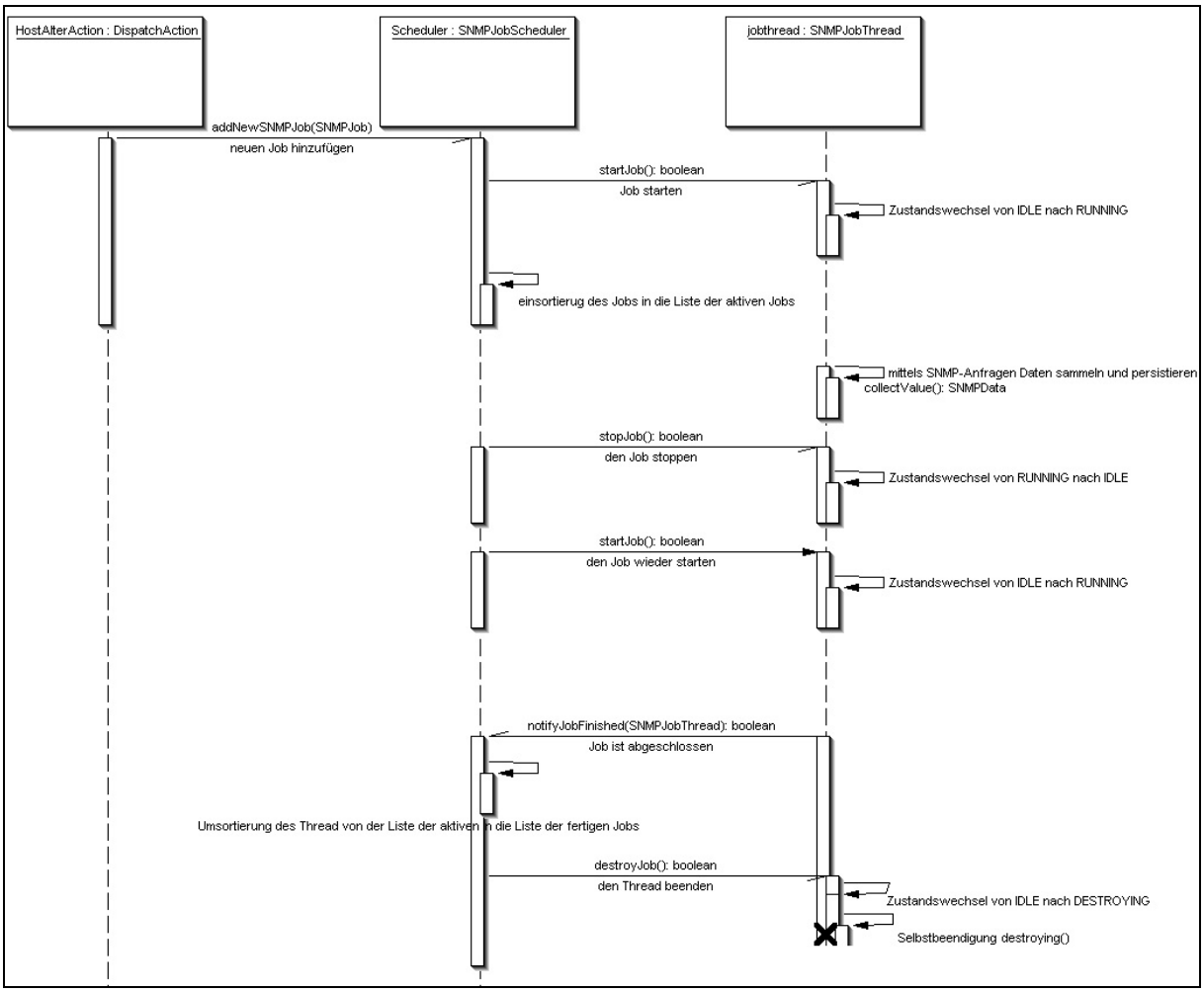

Abbildung 32: Sequenzdiagramm Joberstellung und -verarbeitung

Eine View-Komponente möchte einen Job erstellen, der zur Verarbeitung freigegeben werden soll. In dem Beispiel der Abbildung ist es eine Action-Klasse des Struts- Frameworks.
Der Scheduler veranlasst eine Instanziierung eines verarbeitenden Threads und startet diesen. Dabei sortiert er ihn ein eine Liste von gestarteten Threads ein.
Der Thread vollzieht seine Arbeit, kann aber vom Scheduler gestoppt und wieder gestartet werden. Sobald der Thread seine Arbeit erledigt hat, meldet er dies dem Scheduler, der diesen Job von den Laufenden in die abgeschlossenen Jobs einsortiert und den Thread auffordert sich zu beenden.
Der Thread beendet sich und gibt genutzte Ressourcen wieder frei.
Aus dem Zusammenspiel zwischen Scheduler und Thread ergibt sich für den Job ein etwas umfangreicherer Zustandsautomat der sich mit fünf Zuständen wie folgt darstellt.

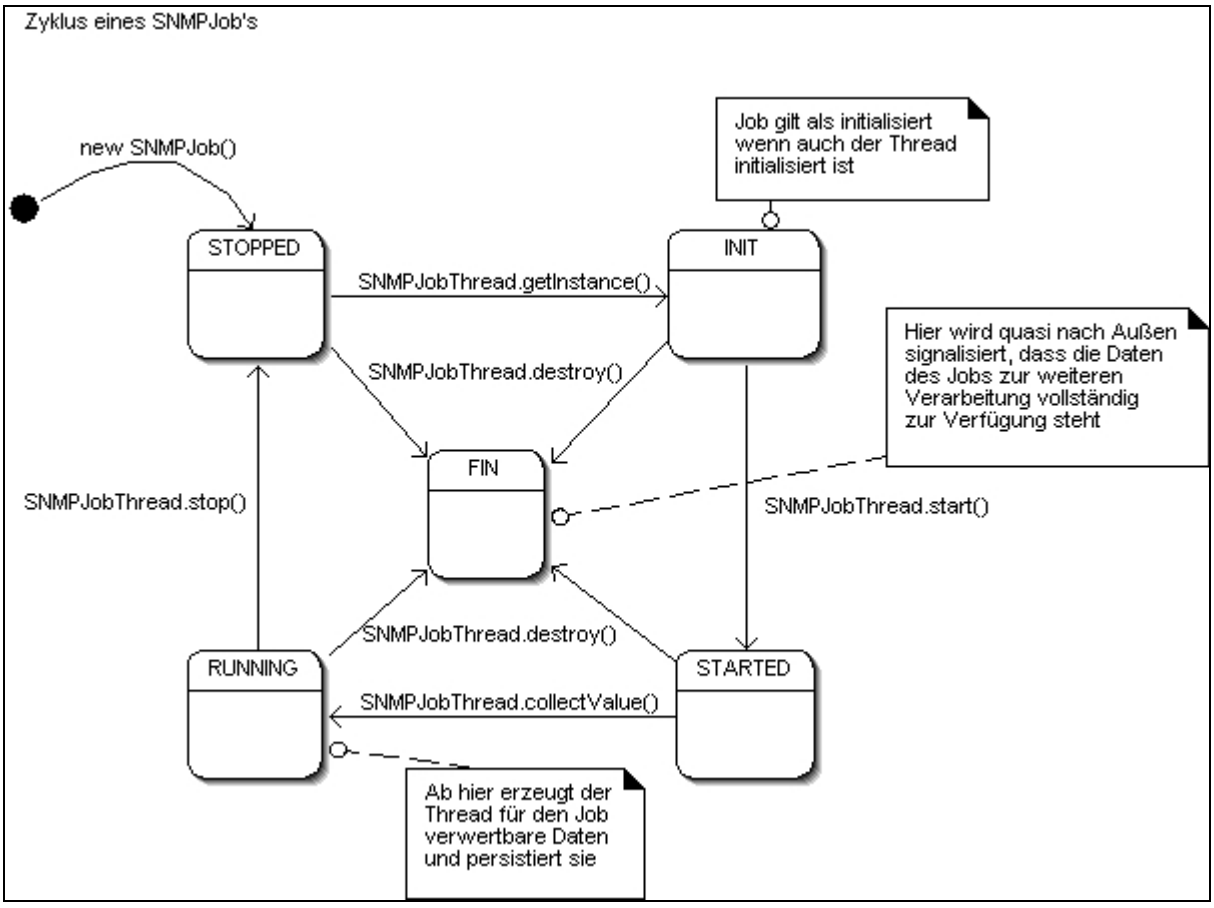

Abbildung 33: Zustandsautomat eines SNMP-Jobs

Zustandsänderungen des Jobs werden allein durch seine Erschaffung und dem bearbeitenden Thread durchgeführt. Hierbei werden die Zustände für den äußeren Betrachter fast ablesbar.

Wenn ein Job durch den new()-Operator erschaffen wird, befindet er sich im Zustand *stopp*, dies ist gleichbedeutend damit, dass er noch nicht dem Scheduler übergeben worden ist. Findet man einen Job im Zustand *init* auf, wurde ein Thread mit diesem erzeugt, allerdings noch nicht gestartet. Geschieht dies letztendlich, liegt der Zustand *started* vor und sobald Daten erhoben werden, erkennt man dies am Zustand *running* des Jobs. Wenn alle Arbeiten abgeschlossen sind, erkennen wir dies, indem der Zustand *fin* eingetreten ist. Verwertbare Daten findet man vor, sobald die Zustände *running* oder *fin* im Job vorliegen.

6. Implementierung

Die View-Komponente wurde in dieser Implementierung mit Tomcat, JSP und Servlets in den Grundtechniken entwickelt. Hierbei wurden einige Frameworks für die View genutzt. Das Struts-Framework bot sich an, um Grundmechanismen der MVC-Kultur zu etablieren. Struts wurde mit weiteren Frameworks kombiniert, wie dem Tiles-Framework, zur Erstellung wieder verwertbarer Designs und dem Validator-Framework zur Validierung von Formulareingaben. In vorausgegangenen Unterkapiteln 5.5 und 5.6 wurde schon darauf hingewiesen, dass zwei Packages genutzt worden sind; auf der einen Seite Mibble, um MIB- Strukturen aus MIB-Files zu parsen, und auf der anderen Seite eine Taglibsammlung, die gestattet mittels ihres Renderes Diagramme in Servlets zu integrieren (Cewolf).
Mibble wurde unter der General Public License (kurz: GPL), Cewolf unter Lesser General Public License veröffentlicht. Daraus ergeben sich folgende Unterschiede für die weitere Nutzung auf kommerzieller Ebene:
Die LGPL erlaubt ohne besondere Einschränkungen, dass aus genutzten Bibliotheken kommerzielle Software entsteht. Bei der GPL-Lizenz hingegen besteht die Verpflichtung, dass Software, die mittels GPL-lizensierten Bibliotheken geschrieben worden ist, diese wiederum unter der GPL-Lizenz veröffentlicht. Das bedeutet, dass jedem voller Zugriff auf alle Sourcen gewährt werden muss, und dass er diese auch verändern und weiterentwickeln darf.

Da die Hauptkomponente aus mehreren Threads besteht, ist zum Erhalt von Debug-Informationen ratsam, einen Loggingmechanismus in den Kode zu integrieren. Die Wahl fiel auf das Log4J-Logging. Log4J lässt sich mit einigen Handgriffen in die Tomcat-Umgebung integrieren. Dabei hat man den Vorteil, dass man auch die Logging-Informationen erhält, die aus dem Jakarta-Projekt stammen, da auch das Apache Jakarta Entwicklung ihren Kode mit Log4J-

Einträgen dekoriert haben. In dieser CoVioN-Implementierung wurde die Initialisierung wegen unterschiedlichen Classloadern mittels Startupservlet vorgenommen. Einzelkomponenten wurden mit JUnit-Testfällen im Vorwege getestet.
Die in Punkt 5.7.2 angeführten Managerklassen, die die anfallenden Daten persistieren, wurden als Singelton implementiert.
Singeltons in Java werden in den meisten Fällen so programmiert, dass der Standard-Konstruktor zunächst „private“ deklariert wird. Somit wird erreicht, dass keiner außer der Klasse selbst den New-Operator mit dieser Klasse verwenden kann. Als weitere Maßnahme hält die Klasse ein statisches Objekt von sich selbst, dessen Zugriff über eine statisch Methode wie, z.B. getInstance(), vorgenommen wird. In der statischen Methode wird geprüft, ob die statische Variable „null“ ist. Wenn das der Fall ist, wird der New-Operator aufgerufen und die statische Variable besetzt. Am Ende wird immer die statische Variable nach außen gereicht.
Mit Class.forName kann man somit nicht so ohne weiters arbeiten, um Singelton-Instanzen zu erhalten, da dies einen Standard-Konstruktor voraussetzt, der als public oder im Packageaccess als protected deklariert wird. Es ist jedoch trotzdem möglich über einen kleinen Umweg, an das gewünschte Ziel zu gelangen. Man erstellt ein Wrapperbean, welches einen public Standard-Konstruktor hat, und erledigt den getInstance()- Aufruf in einer Methode dieser Klasse.
Der Zugriff einer nutzenden Klasse auf die Persistenzschicht erfolgt nun standardmäßig über folgende Codezeilen:

```
String nman = StartupParams.getInstance().getApplicationProperty("nodemanagerbean");
INodeManagerBean nmb=null;
INodeManager nodemanager=null;
try {
        nmb = (INodeManagerBean)Class.forName(nman).newInstance();
} catch (Exception e) {
        e.printStackTrace();
}
if(nmb!=null){
        nodemanager = nmb.getManager();
}
```

Um diesen Code nur einmal schreiben zu müssen, bekam ein weiteres Objekt seine Existenzberechtigung: die Klasse Persister, die einfach alle Manager-Instanzen in sich trägt und bei Bedarf nach außen über Zugriffsmethoden zur Verfügung stellt. Somit verringert sich der Zugriff, z.B. des NodeManagers, auf folgende Zeile:

```
INodeManager man = Persister.getInstance().getNodemanager();
```

Sinngemäß sind auch alle anderen Manager für z.B. Snmp-Jobs oder Snmp-Daten mit demselben Workflow implementiert worden.

6.1 Package-Organisation

CoVioN ist in acht Hauptpackages organisiert, welche jeweils Sinnbereiche ihrer Inhalte wiederspiegeln.

de.quit.covion.application
Hier befinden sich Helperklassen, die bei Konvertierungen hilfreich sind, und Klassen, die applikatinsweite Verarbeitungen vornehmen, so z.B. StartupParams, welche die Konfigurationsdaten der covionconf.xml zur Verfügung stellt.

de.quit.covion.data
Hier sind alle Beans angeordnet, welche applikationsnahe Daten beinhalten, wie SnmpData, Nodes und SnmpJob. Beans, die einem Framework unterliegen, wie z.B. Formbeans von Struts, gehören in ein Sub-/Package von de.quit.covion.view.

de.quit.covion.data.pipes
Diesem Package sollen alle rohdatenverändernden Klassen zugeordnet werden. Dies sind Klassen, die die SNMP-Rohdaten der Abfragen modifizieren oder zum Schluss auf ein Format zuschneiden, die von weiteren Systemen dann zur Weiterverarbeitung genutzt werden können. Hier befindet sich z.B. CovionDatasetProcessor, die SnmpData für einen Renderer vorbereitet, der aus den Daten Graphiken erstellt.

de.quit.covion.mib
Alle Klassen, die zur Verarbeitung von Mib-Dateien benötigt werden, wird man in diesem Package finden.

de.quit.covion.net
Hier befinden sich Klassen, die für die Ausführung einer Netzwerkaktivität benötigt werden. So z.B. eine Pingklasse, mit der man Netzwerkknoten pingen kann.

de.quit.covion.persistence
In diesem Package befinden sich alle Managerimplementationen, die sich um die Persistenz der Applikationsdaten kümmern. Sollen abweichend von der Standardimplementation andere Datensenken implementiert werden, wird empfohlen diese diesem Package zuzuordnen.

de.quit.covion.process
Alle Klassen, die für die Steuerung der Applikation zuständig sind und in diesem Sinne eine Verarbeitung vornehmen, sind diesem Package zugeordnet. Hier befinden sich der SnmpJobScheduler und die SnmpJobThreads.

Die nun folgenden Packages sind bereits losgelöst vom Systemkern. Sie dienten entweder dem Testen von Komponenten oder aber der Realisierung der View-Komponente.

de.quit.covion.test
Dieses Package beinhaltet Hacks und Tests.

de.quit.covion.view.x
Unter diesem Package sind alle Actionklassen, Form-Beans, Customtag-Klassen in Unterpackages organisiert. Actionklassen und Form-Beans sind notwendig für die Verarbeitung der MVC Kultur mittels Struts-Framework, die Customtags vereinfachen das Schreiben von JSP-Seiten und verringern den Gebrauch von Skiptlets innerhalb JSP's.

6.2 Klassenbeschreibungen

Das Herzstück von Covion stellt sich in der Interaktion von Scheduler und den SNMP- Threads dar, da hier die essentiellen Zustandsautomaten implementiert sind, die die Applikation steuern. Das Statechart seitens des Threads wurde schon in Punkt 5.7.4 vorgestellt und flüchtig beschrieben.
Die Run()-Methode von JobThreads enthält folgenden Code:

```
public void run() {
  if(log.isDebugEnabled())
```

```
    log.debug("SNMPJobThread.run: Bin in run()");
  while(!this.isInterrupted()){
    long now = Calendar.getInstance().getTimeInMillis();
    Calendar start = Calendar.getInstance();
    start.setTime(this.job.getStarttime());
    Calendar end = Calendar.getInstance();
    end.setTime(this.job.getEndtime());

    switch(this.getZustand()){
      case SNMPJobThread.IDLE:
        SNMPJobThread.yield();
        break;
      case SNMPJobThread.RUNNING:
        if(now>start.getTimeInMillis() && now<end.getTimeInMillis()){
          SnmpDataPersisterImpl.getInstance().addSNMPData(this.collectValue());
          SNMPJobThread.yield();
          try {
            SNMPJobThread.sleep(job.getIntervall());
          } catch (InterruptedException e) {
            e.printStackTrace();
          }
        }else if(now>end.getTimeInMillis()){
          if(this.stopJob()){
            if(log.isDebugEnabled())
              log.debug("SNMPJobThread.run(): Job ist fertig");
              this.sched.notifyJobFinished(this);
            }
          }
        break;
      case SNMPJobThread.DELETING:
        this.destroying();
        this.interrupt();
        SNMPJobThread.yield();
        break;
    }
  }
return;
}
```

Zunächst werden die Calendar-Objekte mit den Werten des Startzeitpunktes und des Endzeitpunktes aus dem Job erstellt. Im Switch-Case-Konstrukt wird der derzeitige Status des Threads abgefragt und gegen die unterschiedlichen Cases geprüft. Ist der Thread IDLE, so kann dieser die Rechenzeit an einen anderen Thread weitergeben und das Switch-Case-Konstukt mit der break-Anweisung verlassen. Im Fall von DELETING muss der Thread alle genutzten Ressourcen freigeben; dies geschieht in der destroying()-Methode. Mit der interrupt()-Anweisung

des Threads wird die While-Schleife unterbrochen, so dass das Ende der run()-Mehtode erreicht ist und der Thread sich beendet. In Falle vom Zustand RUN wird zunächst überprüft, ob die Startzeit bereits und die Endzeit dagegen noch nicht überschritten ist. Dann ist das Zeitfenster erreicht, in dem der Thread seine Arbeit verrichten soll. Mit dem Methodenaufruf von collectValue() werden SNMP-Anfragen an den Node gesendet und mit der Response SNMPData Objekte erzeugt, die über den Manager persistiert werden. Die Rechenzeit wird wieder an andere threads abgegeben mit yield() und der Thread legt sich für die Dauer des Polligintervalls schlafen. Sollte die Endzeit des Zeitfensters, in der der SNMP-Job zu verrichten ist, abgelaufen sein, stoppt sich der Thread von selbst und meldet dem Scheduler, dass Seine Arbeit abgeschlossen ist. In der notifyJobFinished()-Methode des Schedulers wird die destroy()-Methode des Threads aufgerufen, in dem dieser seinen Zustand auf DELETING und den Zustands des Jobs auf FIN verändert. Im nächsten Schleifendurchlauf tritt der oben beschriebene Selbstbeendigungsmechanismus ein und gibt vorher alle Ressourcen frei. Wenn man sich im Einzelnen die Methoden anschaut, die eine Veränderung der Stati zur Folge haben, erkennt man, dass diese asynchron aufrufbaren Methoden alle nach einem einheitlichen Schema aufgebaut sind:

```
public boolean destroyJob(){
  if(log.isDebugEnabled())
    log.debug("SNMPJobThread: Eintritt in destroyJob() Zustand ist"
        +SNMPJobThread.STR_ZUSTAND[this.zustand]);
    boolean isOK=false;
    switch(this.getZustand()){
      case SNMPJobThread.IDLE:
        this.setZustand(SNMPJobThread.DELETING);
        this.job.setStatus(SNMPJob.FIN);
        if(SnmpJobManagerImpl.getInstance().updateJob(this.job)){
          if(log.isDebugEnabled())
            log.debug("ThreadJob verändert Jobzustand:"+this.getJob());
        }
        isOK=true;
        break;
      case SNMPJobThread.RUNNING:
        this.setZustand(SNMPJobThread.DELETING);
        this.job.setStatus(SNMPJob.FIN);
        if(SnmpJobManagerImpl.getInstance().updateJob(this.job)){
          if(log.isDebugEnabled())
            log.debug("ThreadJob verändert Jobzustand:"+this.getJob());
        }
        isOK=true;
        break;
```

```
      case SNMPJobThread.DELETING:
        if(log.isWarnEnabled())
          log.warn("SNMPJobThread: stopJob() angefordert, obwohl Zustand schon DELETING ist");
        break;
    }
  return isOK;
}
```

Wie man erkennt, werden in den statusverändernden Methoden keinerlei teure Operationen ausgeführt, um den Zustandswechsel möglichst schnell zu vollziehen. I/O wird nur aus der run()-Methode heraus unternommen. Die collectValue()-Methode hat I/O in Form von Netzwerkinteraktivität:

```
private SNMPData collectValue(){
  if(this.job.getStatus()!= SNMPJob.RUN){
    this.job.setStatus(SNMPJob.RUN);
    if(SnmpJobManagerImpl.getInstance().updateJob(this.job)){
      if(log.isDebugEnabled())
        log.debug("ThreadJob verändert Jobzustand:"+this.getJob());
    }
  }
  SnmpManager manager = null;
  Calendar timestamp =null;
  int value=0;
  String ip = SNMPJobThread.getIpToString(this.job.getNode().getIp());
  try {
    manager= SnmpManager.createSNMPv1(ip,161,this.job.getNode().getPublicKey());
  } catch (SnmpException e) {
    e.printStackTrace();
  }

  if(manager!=null){
    SnmpResponse res = null;

    try {
      res = manager.get(this.job.getOid());
    } catch (SnmpException e1) {
      e1.printStackTrace();
    }
    if(res!=null){
      String val = res.getValue(this.job.getOid());
      timestamp = Calendar.getInstance();
      value=Integer.parseInt(val);
      manager.destroy();
    }
  }
```

```
  return new SNMPData(this.job,timestamp.getTimeInMillis(),value);
}
```

Die Methode illustriert, wie einfach die Verwendung von mibble in Bezug auf SNMP-Anfragen ist. Man instanziiert einen SnmpManger über die Zeile

```
SnmpManager manager = SnmpManager.createSNMPv1(String IP-Adresse,int Portnummer,String
                                                                          Communitystring);
```

und kann mit ihr nun einen SNMP-GET Request starten. Gleichzeitig erhält man den Response für den Request.

```
SnmpResponse res = manager.get(String oid);
String value = res.getValue(String oid);
```

Value enthält nun den Wert, der sich hinter der SNMP-Instanz der MIB verbirgt. Dieser muss zunächst noch geparsed werden. Verbunden mit einem Zeitstempel werden dann SNMPData-Objekte erschaffen.

6.4 Implementierungsbeispiel mit Java Server Pages (JSP) und Servlets

Wie bereits erwähnt, wurde bei dieser Implementation das Struts Framework genutzt um MVC zu modellieren. Struts baut darauf auf, dass alle Requests durch einen Frontcontroler geleitet werden. Dieser delegiert anhand von XML-Konfigurationen den Request an entsprechende Action-Klassen weiter, die wiederum den Request bearbeiten, um dann durch eine Weiterleitung zu einer View (meist JSP-Datei) vorzunehmen. Da es sich als recht langwieriges Unterfangen herausstellt jede JSP-Seite von Grund auf neu zu schreiben, wurde ein weiteres Framework in Kombination genutzt; das Tiles-Framework.
Tiles definiert in der tiles-def.xml unterschiedliche Zusammenstellungen von JSP-Dateien und verleiht dieser Zusammenstellung einen Namen. Sobald Struts Tiles als Plugin hinzugeführt worden ist, reagiert Struts bei einer Weiterleitung auf die Namensbezeichner, die in der tiles-def.xml definiert wurden, um in der struts-config.xml Ihren Einsatz zu finden.

Beispiel:

```
<tiles-definitions>
  <definition name="covstdlayout" path ">
    <put name="title" value="CovioN - Correalation Viewer on Networkparts"/>
    <put name="header" value="/shapes/header.jsp" type="page"/>
    <put name="footer" value="/shapes/footer.jsp" type="page"/>
    <put name="mainmenu" value="/shapes/mainmenu.jsp" type="page"/>
    <put name="content" value="/index/home.jsp" type="page"/>
  </definition>
  <definition name="mibbrowser" extends="covstdlayout">
    <put name="content" value="/jobs/mibbrowser.jsp" type="page"/>
  </definition>
…
<tiles-definitions>
```

Ausschnitt aus tiles-def.xml

Die erste Definition beschreibt ein Standardlayout, welches bestimmte Felder in sich definiert hat.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ page language="java" contentType="text/html; charset=iso-8859-1" %>
<%@ taglib uri="/tags/struts-tiles" prefix="tiles"%>
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
  <title><tiles:getAsString name="title"/></title>
  <link rel=stylesheet href="<%=request.getContextPath()%>/style/style.css" type="text/css">
</head>
<body>
<table width="1024" border="0" cellspacing="0">
        <tr>
                <td><tiles:insert attribute="header"/></td>
        </tr>
        <tr>
                <td><tiles:insert attribute="mainmenu"/></td>
        </tr>
        <tr>
                <td><tiles:insert attribute="content"/></td>
        </tr>
        <tr>
                <td><tiles:insert attribute="footer"/></td>
        </tr>
</table>
</body>
</html>
```

In der Datei covstdlay.jsp sind die Tags, wie <tiles:insert attribute='header'/> Definitionen für Ersetzungsverfahren von abgeleiteten Tiles, wie z.B. die Tilesdefinition mibbrowser, die sich von cobstdlayout ableitet und nur das Tiles-Attribute content mit einer anderen JSP-Seite laden lässt. In einer Strutsdefinition werden nunmehr Weiterleitungen aufgrund der vergebenen Namen der Tilesdefinitionen geschaltet.

```
…
<form-bean name="FormSelMib" type="org.apache.struts.validator.DynaValidatorForm">
    <form-property name="mib" type="java.lang.String"/>
    <form-property name="method" type="java.lang.String"/>
</form-bean>
…
<action-mappings>
    <action forward="covstdlayout" path="/home"/>
…
    <action input="usemib" name="FormSelMib" parameter="method" path="/mibbrowser"
       scope="request" type="de.quit.covion.view.actions.ToMibbrowserAction" validate="false">
       <forward name="success" path="mibbrowser"/>
    </action>
  </action-mappings>
…
```

Ausschnitt aus struts-conf.xml

Dieses Beispiel beschreibt eine ActionMapping, welches ein Formular zur Verarbeitung bekommt, mit dem Namen FomSelMib vom Typ DynaValidatorForm. Der Ursprung dieses Requests soll von dem Tile mit dem Namen usemib stammen, das abgesendete Formular hat die Entitäten method und mib jeweils vom Typ String. Diese Formulareingaben werden nun vom Validator-Framework übernommen und auf ihre Korrektheit überprüft. Jedes Formular, welches einer Überprüfung seiner Felder bedarf, erhält einen Eintrag im Konfigutationsfile des Validator-Frameworks validation.xml, welches in dem oben aufgeführten Beispiel so aussieht.

```
…
<form name="FormSelMib">
    <field property="mib" depends="required">
      <arg0 key="label.mib.mib"/>
    </field>
</form>
…
```

Ausschnitt aus validation.xml

Der Validator enthält Standardabfragen zu fast allen Standardfällen und hat zudem noch den Vorteil, eine clientseitige sowie serverseitge Validierung vorzunehmen. Wünschen wir eine clientseitige, vom Browser verursachte Prüfung von Feldern geschieht dies mittels generiertem Javascript, der im Formtag mit integriert werden muss. Folgende Zeile entspricht einer clientseitgen Validierung; integriert im Formtag des Formulars:

```
<html:form action="/hostsactiondo" onsubmit="return validateHostAlterForm(this);">
```

Hierbei ist es wichtig zu erwähnen, dass die Javascriptmethode nicht vom Entwickler geschrieben werden muss; die Methode wird automatisch vom Validator-Framework generiert. Dabei lässt sich auch bei komplizierteren Validierungen eingreifen und eigene Javascripts bei Bedarf einbinden. Als letztes Glied der MVC-Architektur fehlt noch die Actionklasse, die aufgrund des Forms entscheidet, zu welchem Tile ein erfolgreiches Absenden des Formulars weiter geleitet wird. In der struts-config.xml konnte man nur einen Fall erkennen, der mit success beschrieben wurde. In Standardfällen von Formulareingaben würden hier mehrere Fälle beschrieben werden, die zu unterschiedlichen Tiles weitergeleitet werden können. Die beschriebene Actionklasse stellt sich wie folgt dar:

```
public class ToMibbrowserAction extends DispatchAction {

/* forward name="success" path="mibbrowser" */
private final static String SUCCESS = "success";

  public ActionForward tomibbrowser(ActionMapping mapping,
      ActionForm form,
      HttpServletRequest request,
      HttpServletResponse response){

    if(log.isDebugEnabled()){
      log.debug("====================>  Forward nach "+ToMibbrowserAction.SUCCESS);
    }
    return mapping.findForward(ToMibbrowserAction.SUCCESS);
  }
}
```

Die Actionklasse sieht durch die Verwendung des Validatorframeworks relativ kurz aus. Besonders Entwickler, die sich schon einmal mit Struts beschäftigt haben, werden eine Validierungsmethode vermissen. Da früher jedoch oftmals immer wieder Standarduntersuchungen aller Felder stattgefunden haben, ähnelten sich auch die validate()-

Methoden; manche waren teilweise doppelt und dreifach in einem Projekt vorhanden. Die Implementierung sämtlicher validate()-Methoden wird durch die Verwendung des Validator-Frameworks eingespart.

6.1 Implementierungsbeispiel des Persistenz-Interface mit db4Objects und MySQL

Mit welchen Mitteln man die Persistenz von den Objekten Node, SNMPJob, Anwender usw. herbeiführt, sollte zunächst für den Core nicht von Belang sein. Die Schritte, die man unternehmen muss, um auf eine neue Persistenzschicht umzuschwenken ist relativ einfach. An dem Beispiel einer Implementierung für Nodes sollen kurz zwei Implementierungen in MySQL und db4objects vorgestellt werden.
Zunächst ein Blick auf die zu implementierenden Interfaces INodeManager und INodeManagerBean:

```
public interface INodeManagerBean {
  public INodeManager getManager();
}
```

Interface INodeManagerBean

```
public interface INodeManager {
  public Node getNodeById(int id);
  public Node getNodeByIp(byte[] ip);
  public Node getNodeByIp(String ip);
  public Node getNodeByIp(InetAddress ip);
  public Node[] getAllManagedNodes();
  public Node[] getAllNodes();
  public boolean addNode(Node node);
  public boolean updateNode(Node node);
  public boolean deleteNode(Node node);
  public boolean deleteAllNodes();
  public boolean dbOk();
  public Node[] sortById(Node[] nodes);
  public Node[] sortByIP(Node[] nodes);
  public Node[] sortByName(Node[] nodes);
}
```

Interface INodeManager

In Punkt 5.7.2 vorweggenommen, soll trotzdem noch einmal die Problematik ins Gedächtnis zurückgerufen werden, dass Singeltonimplementierungen unter Java nur mittels Umwegen

konfigurierbar erstellt werden können. Deshalb bedient sich diese Implementierung eines Beans, welches mittels Standardkonstruktors konfigurierbar erstellt werden kann, und darüber hinaus die Möglichkeit hat, eine static deklarierte Methode eines Singeltons auszuführen, und somit den Singelton zu instanziieren.
Folgende Zeilen stellen die Implementierung für db4objects, einem objektorientierten Ansatz der Persistenzschicht dar. Es sollen dabei Snippets der Methode addNode(Node n) beispielhaft vorgestellt werden.

```
public class NodeManagerBean implements INodeManagerBean {

  private INodeManager manager;

  public NodeManagerBean() {
    this.manager= NodeManagerImpl.getInstance();
  }

  public INodeManager getManager(){
    return this.manager;
  }
}
```

Implementierung von INodeManagerBean

Dem MySQL-Gegenstück liegt in groben Zügen dieselbe Implementierung zu Grunde, lediglich mit dem Unterschied, dass eine andere Klasse im Konstruktor aufgerufen wird. Es wird an dieser Stelle deshalb darauf verzichtet, diese noch einmal darzustellen.

```
public class NodeManagerImpl implements INodeManager {
  private static transient Log log = LogFactory.getLog(NodeManagerImpl.class);

  public static NodeManagerImpl manager= null;

  public NodeManagerImpl(){}

  public static NodeManagerImpl getInstance(){
    if(manager==null) {
      if(log.isDebugEnabled()){
        log.debug("Nodemanagerinstanz erzeugt!");
      }
      NodeManagerImpl.manager=new NodeManagerImpl();
    }
    return NodeManagerImpl.manager;
}
…
```

```
  public boolean addNode(Node node) {
    ObjectContainer db=DBServer.getInstance().srv();
    if(log.isDebugEnabled()){
      log.debug("addNode(): Client geöffnet");
    }
    boolean opCorrect = false;
    try {
      if(!this.nodeExist(node)){
        int id = getNextId();
        if(log.isDebugEnabled())
          log.debug("NodeManagerImpl.addNode(): als nächste ID ist "+id+" vorgesehen
                  worden!");
        node.setId(id);
        db.set(node);
        opCorrect=true;
        db.commit();
        DBServer.getInstance().closeClient(db);
        db= null;
        if(log.isDebugEnabled())
          log.debug("NodeManagerImpl.addNode() Node sollte hinzugefügtworden
                  sein=["+node.getId()+"] "+node.getName());
      }
    }
    finally {
      if(db!=null){
        DBServer.getInstance().closeClient(db);
        db= null;
      }
    }
  return opCorrect;
}
```

Ausschnitt der addNode(Node n)-Methode aus NodeManagerImpl Klasse

Für die Umsetzung von db4Object wurde ein Server geschrieben, von dem ein Client einen ObjectContainer beziehen kann. Über den ObjectContainer lassen sich unterschiedliche Methoden aufrufen, so z.B. die set()-Methode, die dem Persistieren eines Objekts gleichkommt. Es ist möglich, die Datenbasis mit QBE (query by example) abzufragen. Das bedeutet, dass man ein Dummy-Objekt mit den gewünschten Objektinhalten vorbereitet und mittels der get()-Methode in der Datenbasis mit den vorhandnen Objekten vergleicht. Übereinstimmende Objekte werden in einem ObjectSet zusammengestellt und zur weiteren Verarbeitung nach Außen geleitet.

Eine weitere Möglichkeit Objekte aus der Datenbasis zu erhalten, bietet ein alternatives Querying. Beispiel (ein Node mit der ID 3):

```
Query q = db.query();
q.constrain(Node.class);
q.descend("id").constrain(new Integer(3));
ObjectSet result=q.execute();
if(result.size()==1){
  nd = (Node) result.next();
}
```

Beispiel mittels Query-Methodik

```
Node n = new Node();
n.setId(3);
ObjectSet result=db.get(n);
if(next.size()==1)
  Node dbnode = (Node)result.next();
```

Beispiel mittels QBE-Methodik

Die Implementierung auf Grundlage von MySQL als Datenbasis könnte in Bezug der Methode addNode(Node n) so aussehen:

```
…
public boolean addNode(Node node) {
  if(this.getNodeByIp(node.getIp())!=null){
    return false;
  }
  boolean isok=false;
  StringBuffer sql = new StringBuffer();
  sql.append("INSERT INTO node SET ");
  sql.append("ip=").append(DataToStringHelper.ip2int(node.getIp())).append(", ");
  sql.append("name='").append(node.getName()).append("', ");
  sql.append("publickey='").append(node.getPublicKey()).append("', ");
  sql.append("privatekey='").append(node.getPrivateKey()).append("', ");
  sql.append("red=").append(node.getRed()).append(", ");
  sql.append("orange=").append(node.getOrange()).append(", ");
  sql.append("green=").append(node.getGreen());

  Connection con = null;
  Statement dbstatement = null;
  ResultSet res = null;
  try {
    con = DBHelper.getConnection();
    dbstatement = con.createStatement();
    if(dbstatement.executeUpdate(sql.toString())==1){
      isok=true;
    }
    con.close();
```

```
  } catch (SQLException e1) {
    e1.printStackTrace();
  }finally{
    DBHelper.free(con);
    if (dbstatement != null) {
      try {
        dbstatement.close();
      } catch (SQLException sqlEx) { // ignore }
        dbstatement = null;
    }
  }
}
return isok;
}
…
```

Ausschnitt addNode(Node n) aus MyNodeManager

Die genutzte Klasse DBHelper trägt eine Implementierung eines DBConnection-Pools in sich. Sie stellt mit der Methode getConnection() ein Connection-Objekt zur Verfügung und übernimmt gleichzeitig die Rückmeldung an den Connection-Pool mit der Methode free(Connection con).

7. Durchgeführte Testszenarien an CoVioN

CoVioN wurde mehreren Tests von Einzelkomponenten über JUNIT-Testfälle und normalem debugging im Package de.quit.covion.test mittels Testklassen unterzogen.
Zum Testen der GUI, bzw. Viewkomponente, wurden dem System das Loopbackinterface, sowie ein VoIP-Gateway als Hosts hinzugefügt.
Ein Test über den ausgehenden Verkehr des Interfaces 2, des Testrechners mit der IP-Adresse 192.168.0.3 kam ohne kontinuierliche Mittelwertbildung zu folgender Graphik:

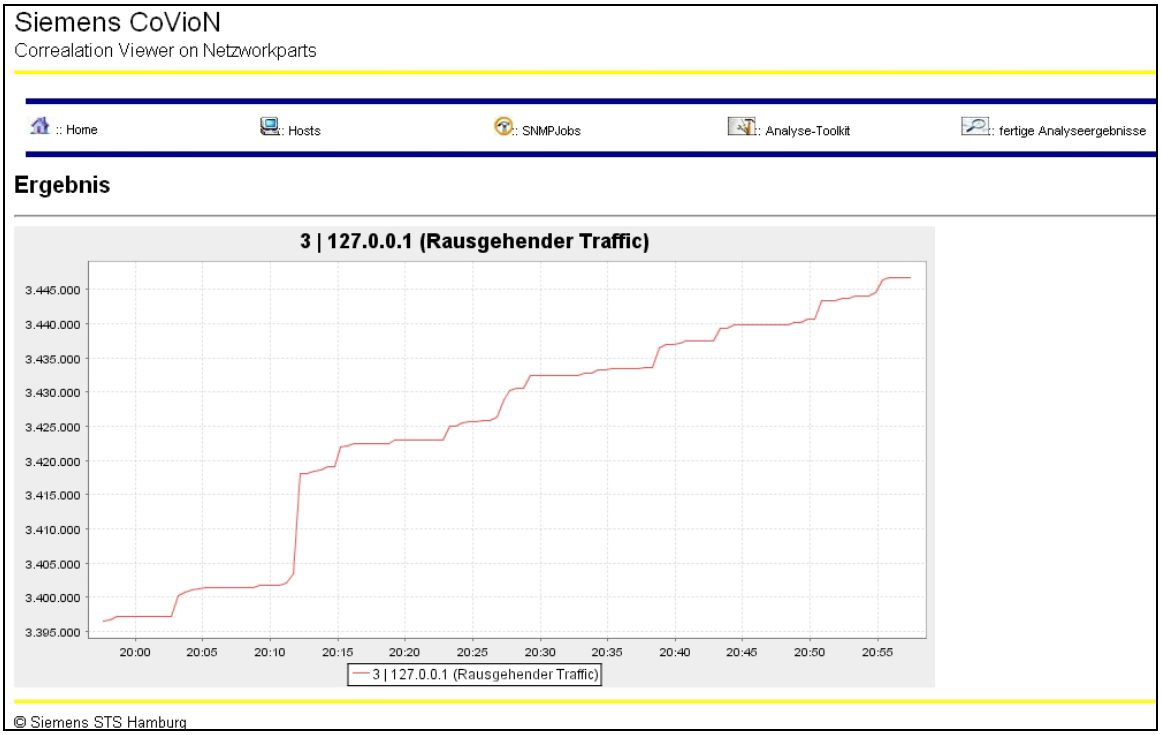

Abbildung 34: Ergebnischart Ausgehender Traffic ohne Mittelwertbildung

Mit kontinuierlicher Mittelwertbildung über die letzen 10 Werte erschließt sich folgendes Chart:

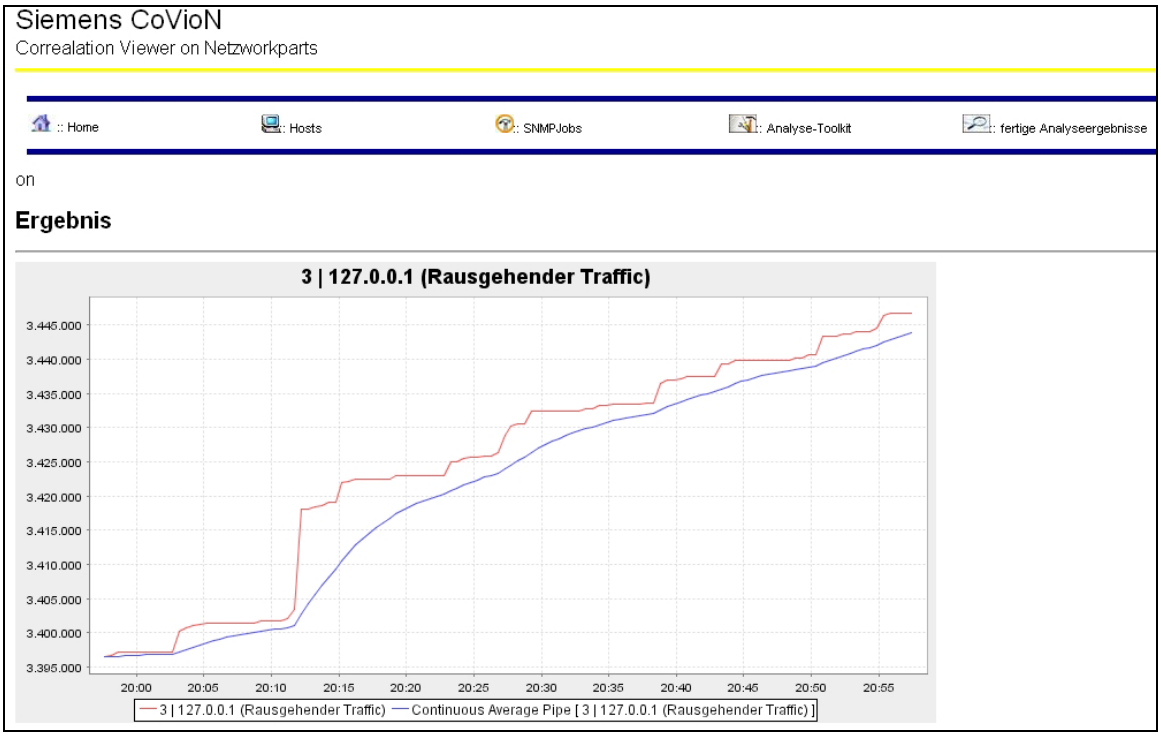

Abbildung 35: Ergebnischart Ausgehender Traffic mit kontinuierlicher Mittelwertbildung

Die Bezeichnungen der Kurven werden automatisch aus Kombinationen von beteiligten Nodes und Pipes gebildet.

8. Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Konzept entwickelt, mit dem es möglich ist auf relativ einfache Art und Weise Datenerhebungen auf Basis von SNMP zur Analyse von Netzwerksystemen zu erheben. Die auf Diagrammen basierenden Resultate können dabei behilflich sein, Phänomene in einem Netzwerk zu untersuchen und Fehlerquellen schneller zu lokalisieren. Hierbei wurde auf die Möglichkeit Korrelationen unterschiedlicher Systemparameter über mehrere System zu erstellen eingegangen und Möglichkeiten aufgezeigt, wie man hierbei standardisierende Vorgehensweisen als Szenarien für spätere Untersuchungen vorhalten, bzw. persistieren kann. Ein Einblick in die Programmierung sollte einen weiteren Einblick in das Projekt verschaffen und das Verständnis des Kernsystems von CoVioN vertiefen. Hierbei wurde auf die Austauschfähigkeit der Persistenzschicht von CoVioN eingegangen und welche Schritte unternommen werden müssen, um dem Kern eine andere Persistenzschicht über Konfiguration mittels XML zur Verfügung zu stellen. Im Zuge einer exemplarischen Benutzeroberfläche mit webbasierten Javatechniken, entstand ein Ansatz für einen MIB-Browser, der ohne Einflüsse von Frameworks zur Erstellung von Oberflächen, wie SWING, AWT oder SWT funktioniert. Während der Erstellung der Weboberfläche wurden Custom-Tags erstellt, deren Implementierungen im Sourcecode auf der CD im Anhang zu entnehmen und einzusehen sind.
Auch wurde auf die Standards eingegangen, die bei einer Einführung von VoIP in Netzwerken zum tragen kommen, um die Qualitätsansprüche an der Qualität von herkömmlicher PSTN-Technik bemessen werden. Dies führte durch Erörterungen und Beleuchtungen von QoS in den Schichten L2 und L3, die verinnerlichen sollten, mit welchen Problemen paketvermittelnde Dienste zu kämpfen haben und wie sich Garantien bezüglich Sprachqualität in Techniken umsetzen lassen.
CoVioN ist allerdings noch in einem frühen Entwicklungsstadium und bedarf noch an weiterer Entwicklung bezüglich einer Verwaltung von Data-Pipes, einer Implementierung eines JNI-Pings und das Absenden von ad-hoc SNMP-Abfragen über den MIB-Browser. Auch wurde nur das Konzept eines Trap-Managers dargelegt, sowie ein Vorschlag unterbreitet, wie Szenarien wieder verwertbar persistiert werden können.
Da Mibble unter der GPL veröffentlicht wurde, wird auch dies Konzept unter der GPL hiermit veröffentlicht und somit der Wunsch nach weiterer Entwicklung unterbreitet.

A Literaturverweise und Links

[1]
Niclas Ek: IEEE 802.1 P,Q - QoS on the MAC level. URL http://www.tml.hut.fi/Opinnot/Tik-110.551/1999/papers/08IEEE802.1QosInMAC/qos.html; Helsinki University of Technology – vom 24.04.1999

[2]
Cewolf-charting the net URL. http://cewolf.sourceforge.net/new/

[3]
Cerderberg Per: Mibble::MIB Parser. URL http://www.mibble.org/

[4]
Stallings William: SNMP, SNMPv2, SNMPv3 and RMON1 and 2, 3. Auflage, Addison-Wesley Verlag, April 2004 – ISBN 0-201-48534-6

[5]
M. Rose, K. McCloghrie: RFC-1155 – “Structure and Identification of Management Information for TCP/IP-based Internets“ URL http://www.ietf.org/rfc/rfc1155.txt, Mai 1990

[6]
M. Rose, K. McCloghrie: RFC-1213 – “Structure and Identification of Management Information for TCP/IP-based Internets: MIB-II“ URL http://www.ietf.org/rfc/rfc1213.txt, März 1991

[7]
M. Rose: RFC-1215 – “A Convention für Defining Traps for use with the SNMP“ URL http://www.ietf.org/rfc/rfc1215.txt, März 1991

[8]
Audio Video Transport Working Group (H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson): RFC-1889 – “RTP: A Transport Protocol for Realtime Applications” URL http://www.ietf.org/rfc/rfc1889.txt, Januar 1996

[9]
SNMPv2 Working Group (J. Case, K. McCloghrie, M. Rose, S. Waldbusser): RFC-1901 – “Introduction to Community-based SNMPv2” URL http://www.ietf.org/rfc/rfc1901.txt, Januar 1996

[10]
SNMPv2 Working Group (J. Case, K. McCloghrie, M. Rose, S. Waldbusser): RFC-1902 – “Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)” URL http://www.ietf.org/rfc/rfc1902.txt, Januar 1996

[11]
SNMPv2 Working Group (J. Case, K. McCloghrie, M. Rose, S. Waldbusser): RFC-1903 – “Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2)” URL http://www.ietf.org/rfc/rfc1903.txt, Januar 1996

[12]
SNMPv2 Working Group (J. Case, K. McCloghrie, M. Rose, S. Waldbusser): RFC-1904 – “Conformance Statements for Version 2 of the Simple Network Management Protocol (SNMPv2)” URL http://www.ietf.org/rfc/rfc1904.txt, Januar 1996

[13]
SNMPv2 Working Group (J. Case, K. McCloghrie, M. Rose, S. Waldbusser): RFC-1905 – “Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)” URL http://www.ietf.org/rfc/rfc1905.txt, Januar 1996

[14]
SNMPv2 Working Group (J. Case, K. McCloghrie, M. Rose, S. Waldbusser): RFC-1906 – “Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)” URL http://www.ietf.org/rfc/rfc1906.txt, Januar 1996

[15]
SNMPv2 Working Group (J. Case, K. McCloghrie, M. Rose, S. Waldbusser): RFC-1907 – “Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)” URL http://www.ietf.org/rfc/rfc1907.txt, Januar 1996

[16]
SNMPv2 Working Group (J. Case, K. McCloghrie, M. Rose, S. Waldbusser): RFC-1908 – “Coexistence between Version 1 and Version 2 of the Internet-standard Network Management Framework” URL http://www.ietf.org/rfc/rfc1908.txt, Januar 1996

[17]
D. Harrington, R. Presuhn, B. Wijnen: RFC-2271 - “An Architecture for Describeing SNMP Frameworks” URL http://www.ietf.org/rfc/rfc2271.txt, Januar 1998

[18]
J. Case, D. Harrington, R. Presuhn, B. Wijnen: RFC-2272 - “Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)” URL http://www.ietf.org/rfc/rfc2272.txt, Januar 1998

[19]
D. Lewi, P. Meyer, B.Steward: RFC-2273 - “SNMPv3 Applications” URL http://www.ietf.org/rfc/rfc2273.txt, Januar 1998

[20]
U. Blumenthal, B. Wijnen: RFC-2274 - “User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)” URL http://www.ietf.org/rfc/rfc2274.txt, Januar 1998

[21]
B. Wijnen, R. Presuhn, K. McCloghrie: RFC-2275 – “View-based Access Control model (VACM) for the Simple Network Management Protocol (SNMPv3)” URL http://www.ietf.org/rfc/rfc2275.txt, Januar 1998

[22]
K. Nichols, S. Blake, F. Baker, D. Black: RFC-2474 – “Definition of the Differentiated Service Field (DS-Field) in the IPv4 and IPv6 Headers” URL http://www.ietf.org/rfc/rfc2474.txt, Dezember 1998

[23]
J. Heinanen, F. Baker, W. Weiss, J Wroclawski: RFC-2597 – “Assured Forwarding PHB Group” URL http://www.ietf.org/rfc/rfc2597.txt, Juni 1999

[24]
M. Baugher, B. Strahm, I. Suconick: RFC-2959 – “Real-time Transport Protocol Management Information Base” URL http://www.ietf.org/rfc/rfc2959.txt, Oktober 2000

[25]
H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson: RFC-3550 – “RTP: A Transport Protocol for Realtime Applications” URL http://www.ietf.org/rfc/rfc3550.txt, Juli 2003

[26]
H. Schulzrinne, S. Casner: RFC-3551 – “RTP Profile for Audio and Video Conferences with minimal Control” URL http://www.ietf.org/rfc/rfc3551.txt, Juli 2003

[27]
M. Wolff, M. Albrecht: “Struts GE-PACKT”, 1. Auflage, mitp-Verlag, 2004, ISBN 3-8266-1431-3

[28]
K. Lipinsky : ”SMP-Protokol” URL http://www.itwissen.info/?ano=01-006368&id=31, DATACOM Buchverlag GmbH, vom 05.07.2005

[29]
D. Harrington, R. Presuhn, B. Wijnen: RFC-3411 – “An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks” URL http://www.ietf.org/rfc/rfc3411.txt, Dezember 2002

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(4) bzw. §25(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Ort, Datum Unterschrift