Institut für Informatik

der Ludwig-Maximilians-Universität München

Fortgeschrittenenpraktikum

Implementierung eines Discovery-Agenten

Datum:	1.3. 2000
Bearbeiter:	Karin Thaens
Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Norbert Wienold

1. EINLEITUNG 4

2. SYSTEMVORAUSSETZUNG 5

2.1. MASA – Mobile Agent System Architecture 5

2.2. Testumgebung 5

2.3. Entwicklungsumgebung 6

3. THEORETISCHE GRUNDLAGEN 7

3.1. LAN-Komponenten 7

3.1.1. Repeater 7

3.1.2. Bridge 7

3.1.3. Router 8

3.1.4. Endsysteme 8

3.2. Topologien 8

3.2.1. Token Ring 8

3.2.2. Ethernet 9

3.3. Werkzeuge 10

3.3.1. DNS 10

3.3.2. NIS 10

3.3.3. SNMP 11

3.3.4. MIB 12

3.3.5. MIB-II 12

3.3.6. DOT1Bridge-MIB 13

3.3.7. RMON 13

3.3.8. PING 14

3.4. Objektorientierte Modellierungstechnik 15

3.4.1. UML 15

4. REALISIERUNG 17

4.1. Theorie und Praxis 17

Das Agentenapplet 18

4.2. Der Programmablauf 22

4.2.1. Die Architektur 22

4.2.2. SnifferAgentApplet 23

4.2.3. SnifferAgentMobileAgent 24

4.2.4. examine netvec.allElements 25

4.2.5. Snmpget 26

4.2.6. Das Bridge-Modul 27

4.2.7. Finale 29

5. AUSBLICK UND ZUSAMMENFASSUNG 29

6. ANHANG 30

7. ABBILDUNGSVERZEICHNIS 32

8. LITERATURVERZEICHNIS 33

1. Einleitung

In der heutigen Zeit, in der fast jeder Arbeitsplatz mit einem Rechner ausgestattet ist, wird die Anforderung eines Netzmanagement bereits in kleineren Firmen laut. Überwachungstools für Netzlast, individuell konfigurierbare Sicherheitsfilter und Ansätze eines Fehlermanagements finden sich in kleinerem Rahmen bereits in vielen Betriebssystemen wieder. Professionelle und kommerzielle Produkte wie HP Open-View oder Sprectrum können zu Managementzwecken die Topologie einer heterogenen Umgebung erforschen, indem sie Systemdaten der einzelnen Komponenten sammeln, diese auswerten und anschließend graphisch darstellen.

Dieses Fortgeschrittenenpraktikum befaßt sich mit einem speziellen Management-Werkzeug. Im Rahmen dieser Arbeit wird ein Agent entwickelt, der sich um die Beschaffung von Topologiedaten in einem heterogenen Netz kümmert. Die Implementierung erfolgt auf objektorientierter CORBA-Basis. Als Laufzeitumgebung für den Agenten wird die lehrstuhlintern entwickelte Agentenumgebung MASA verwendet, die als Schnittstelle zu CORBA dient und sämtliche Verwaltungsaufgaben, die zur Laufzeit anfallen, übernimmt.
Die anschließende graphische Darstellung der Netztopologie wird von einem Zusatzmodul übernommen, daß parallel in einem weiteren Fortgeschrittenenpraktikum entwickelt wird.

Im folgenden möchte ich nun genauer auf die untersuchten Netzstrukturen, die Managementmöglichkeiten und die diesem Agenten zugrunde liegende MASA-Umgebung eingehen, um anschließend die Realisierungsidee und die tatsächliche Implementierung vorzustellen.

2. Systemvoraussetzung

2.1. MASA – Mobile Agent System Architecture

Die an diesem Lehrstuhl entwickelte und ständig verbesserte Laufzeitumgebung für Agenten mit CORBA wurde in Java 1.2 implementiert und ist plattformunabhängig. Auf einem System gestartet, ermöglicht sie die Verwaltung der Agenten über ein Agenten-System-Applet, das über einen, zu jedem Agentensystem gehörenden Webserver dargestellt wird. Aufgaben wie Migration und Kommunikation der Agenten über den Object Request Broker, hier ORBacus 3.1.2 von Object Oriented Concepts, wird über diese Architektur realisiert.
Das durch Java 1.2 neu angebotene Sicherheitskonzept wurde als letztes in MASA integriert. Jeder Agent muß sich nun eindeutig identifizieren können, bevor er sich nach einem Neustart oder einer Migration an einem Agentensystem anmelden darf. Damit soll verhindert werden, daß sich Agenten ohne Autorisierung unerlaubter Weise an einem System anmelden und diesem mutwillig Schaden zufügen können (z.B. durch Datenmanipulation, Mithören, etc.).
Des weiteren benötigt ein Agent, um sämtliche Funktionalität nutzen zu können, eine Schnittstelle zum Benutzer, die durch ein eigenes Applet realisiert wird, das die komplette Steuerung des Agenten übernehmen kann.
(Detailliertere Informationen siehe [ROE98], [HGR99])

2.2. Testumgebung

Zu Testzwecken wurde das gesamte Teilnetz 129.187.214.x in der Öttingenstraße auf Topologieinformationen untersucht.
Große Schwierigkeiten gab es bei der Analyse von Koppelelementen, deren Verwaltung aufgrund von Effektivität und Überschaubarkeit komplett an das Leibniz-Rechenzentrum ausgelagert ist, das sich um das gesamte Münchner Hochschulnetz, ausgenommen der Endsysteme, kümmert. Das Problem dabei ist, daß es sich bei sämtlichen Koppelelementen um Produktivsysteme handelt, d.h. ausgelastete Systeme, deren Konfiguration und Funktionsfähigkeit essentiell für den laufenden Betrieb des Instituts für Informatik sind. Diese für Testzwecke zu benutzen bedeutet nicht nur zusätzliche Belastung für die Systeme, sondern birgt Risiken, die für den Netzverkehr nicht unerheblich sein können. So standen diesem Fortgeschrittenenpraktikum lediglich Rechte für den lesenden Zugriff auf einen Switch zur

Verfügung, anhand dessen die Funktionalität des Discovery-Agenten nur sehr eingeschränkt getestet werden konnte. Deshalb wird im Kapitel „Realisierung“ genauer auf die Funktionalität der einzelnen Module eingegangen, um diese theoretisch zu erklären.

2.3. Entwicklungsumgebung

- Als Entwicklungsumgebung für den Agenten diente JDK Version 1.2 von SUN unter Linux.
- Das Agenten-Applet wurde mit Hilfe von SWING 1.2 und dem NetBean Developper 1.2 von SUN entwickelt.
- Die CORBA-Grundlage für das MASA-System stellte ORBacus 3.1.2 der Firma Object Oriented Concepts dar.
- Die SNMP-Abfragen wurden durch ein Java-Package der Firma Advent Network Management, Inc. realisiert. Um Versionsunabhängigkeit einhalten zu können, wird dieses Paket mittels shell-Skriptes bei jedem Compiliervorgang neu aus seinem Ursprungsverzeichnis in das Agenten-jar-File eingebunden (siehe Anhang).

3. Theoretische Grundlagen

Ziel dieses Fortgeschrittenenpraktikums ist es einen Agenten zu implementieren, der grundlegende Daten für die Erstellung einer grafisch darstellbaren Topologie liefert. Anhand von DNS (Domain Name Service), NIS (Network Information Service), MIBs (Management Information Bases) und weiterer Werkzeuge werden Informationen über das Netz gesammelt, korreliert und strukturiert gespeichert. Zuerst stellte sich die Frage, welche Rohdaten für die Identifizierung von Topologieinformation nötig sind.

3.1. LAN-Komponenten

Um diese Frage zu beantworten, werden nun im folgenden kurz die wichtigsten LAN-Komponenten, die als Topologieinformations-Lieferanten in Frage kommen, beschrieben: (Detailliertere Informationen siehe [KER93], [TAN97])

3.1.1. Repeater

Der Repeater wird der OSI-Schicht 1 (Bitübertragungsschicht) zugeordnet. Er verbindet zwei gleiche LAN-Typen bidirektional miteinander (Multi-Port-Repeater auch mehrere), d.h. das Medium, die Zugriffstechnik und die Protokolle müssen gleich sein. Er kann weder Konvertierungs- noch Filterfunktion übernehmen. Seine Aufgabe ist es lediglich die Signale zu verstärken und somit die geographische Ausbreitung eines LANs zu vergrößern. Der Repeater ist für Endsysteme komplett transparent. Er enthält häufig keine Management-Agenten und scheidet somit als mögliche Informationsquelle aus.

3.1.2. Bridge

Die Bridge wird der OSI-Schicht 2a (Mediumzugriffsschicht) zugeordnet. Sie verbindet Local Area Network Typen (LANs) gleicher oder unterschiedlicher Art miteinander, d.h. Medium und Zugriffsstrategie können unterschiedlich sein (z.B. Koaxialkabel auf Twisted Pair; Token Ring auf Ethernet). Sie besitzt bereits eigene Intelligenz und kann, indem sie Quell- und Zieladresse eines Frames lernt, zur Verkehrsseparierung eingesetzt werden.

3.1.3. Router

Der Router wird der OSI-Schicht 3 (Vermittlungsschicht) zugeordnet. Er verbindet LAN-Typen, die sowohl auf Schicht 1 als auch auf Schicht 2 unterschiedlich sein können. Er kann auch Komponente des Wide Area Networks (WAN) sein. Seine Routingaufgaben beinhalten die „optimale“ Weiterleitung von Paketen über den kürzesten, sichersten oder schnellsten Weg zum Ziel. Dabei wird das Paket, falls das Ziel noch nicht direkt erreichbar ist, über einen oder mehrere dazwischen geschaltete Router geschickt.

3.1.4. Endsysteme

Endsysteme übernehmen keine Koppelelementaufgaben, d.h. sie leiten keine Pakete an Subnetze weiter (PCs, Workstations...).

3.2. Topologien

3.2.1. Token Ring

Einerseits wird die geographische Anordnung eines Kreises/Ringes festgelegt, andererseits wird das Token-Zugriffsverfahren verwendet, auf das hier nicht näher eingegangen wird. Da sich die Token-Ring-Netzkarten von anderen Netzkarten hardwaremäßig unterscheiden und diese Information in der MIB festgehalten wird, kann man anhand dieses Eintrages feststellen, über welche topologische Struktur das gefundene Element kommuniziert. Dieser Unterschied läßt sich wie folgt erklären:

Das Token Ring Zugriffsverfahren zeichnet sich dadurch aus, daß alle auf diesem Ring laufenden Nachrichten aktiv über jede Station geleitet werden. Jede Station muß somit einerseits jeden Frame (Ebene 2!!) überprüfen, ob er an sie gerichtet ist, sich andererseits fast wie ein Repeater um die Weiterleitung kümmern. Wird die Station ausgeschaltet, so muß diese eine Schleifenleitungsüberbrückung (Globe bypass) aktivieren. Die dafür benötigte technische Realisierung muß eine Token Ring Karte unterstützen.

3.2.2. Ethernet

Im Gegensatz zu Token Ring leiten die teilnehmenden Stationen in einem Ethernet-basierten Netz Frames nicht aktiv weiter, sondern hören den laufenden Verkehr auf einem Bus mit. Sobald eine Station auf dem Bus ihre Adresse erkennt, nimmt sie die Nachricht auf (anders als bei Token Ring, bei dem die *sendende* Station „ihre“ erzeugten Frames wieder vom Ring nimmt). Die Station kann einfach ausgeschaltet werden, ohne daß der Bus oder seine Teilnehmer dies registrieren. Nicht nur durch die unterschiedlichen Teilnahmestrategien an einem Netz, sondern auch durch die Zugriffsverfahren unterscheiden sich die Hardwarefunktionalitäten der Ethernetkarte von denen einer Token Ring Karte.

So hat sich bei Ethernet die CSMA/CD-Strategie durchgesetzt, das „Carrier Sense Multiple Access/Collision Detect“- Verfahren, bei dem die Stationen, bevor sie auf den Bus sendend zugreifen, zuerst hören, ob dieser frei ist. Die erste Station, die eine Kollision entdeckt muß ein JAM-Signal versenden um alle anderen darauf aufmerksam zu machen.

3.3. Werkzeuge

Nun werden die Werkzeuge vorgestellt, mittels derer die einzelnen Netzelemente gefunden und genauer untersucht werden können.
Folgende Hilfmittel wurden verwendet:

3.3.1. DNS

Der ***Domain Name Service*** ist ein Verzeichnisdienst, der die Umwandlung von Rechnernamen in IP-Adressen und umgekehrt übernimmt. Dadurch möchte man eine Adressierung durch Rechnernamen ermöglichen, die dem menschlichen Gehirn viel leichter zugänglich ist, als Zahlenreihen.

Zu ARPANET-Zeiten wurde eine zentrale Datei HOSTS.TXT verwendet, die für jeden neu zum ARPANET hinzukommenden Host manuell angepaßt werden mußte. Auf jedem teilnehmenden Rechner lag eine Kopie der HOSTS.TXT-Datei vor. Als 1984 TCP/IP zum Kommunikationsstandard wurde, weitete sich das ARPANET explosionsartig aus, und der Umfang dieser Datei wurde bald gesprengt. So entwickelte sich eine neue Namenshierarchie, deren Adressen aus einer 32-Bit langen Nummer bestehen. Die Namen wurden hierarchisch einer Organisationsstruktur angepaßt, die sich Domains zuordnen lassen, die hauptsächlich administrative Einheiten bilden und auf weitere Domains verweisen – sogenannten Sub-Domains.

[STE93]

Nutzbare Topologie-Information: <u>Zuordnung: IP-Adresse - Rechnername</u>

3.3.2. NIS

Ziel des ***Network Information Service*** ist es dem Benutzer zu ermöglichen, sich in einem verteilten Rechnersystem von jedem Rechner im Netz anzumelden und den Zugriff auf all seine benötigten Ressourcen zu gewähren. Dies setzt voraus, daß eine Konfigurationsdatei über die existierenden Benutzer, deren Paßwörter, die Gruppenzugehörigkeiten und somit eine Zuweisung zu ihren Rechten existiert. Diese auf jedem Rechner zu pflegen und konsistent zu halten ist in großen System sehr aufwendig. So hat die Sun Microsystems,

Inc. den Network Information Service entwickelt, einen zentralen Verzeichnisdienst, dem eine Datenbank zugrunde liegt, die die Konfigurationsdateien ablöst. Ändern sich Daten, so werden diese ausschließlich in der zentralen Datenbank geändert und stehen somit sofort allen NIS-Clients zur Verfügung.
[STE93]

Nutzbare Topologie-Information: Liste der registrierten Hosts im lokalen Netz

3.3.3. SNMP

(Internet-Management: Kommunikationsmodell)

Das Simple Network Management Protocol Version 1 (***SNMPv1***) wurde 1988 als Managementprotokoll für Router, Server, Workstations etc. eingeführt.
Die Spezifikation umfaßt:

- ein Protokoll zum Datenaustausch zwischen Manager und Managed Object,
- einer Formatvorgabe für Managementinformationen und
- allgemeine Informationen über die Festlegung und Definition von Managed Objects.

Sicherheitsmängel, fehlende Manager-to-Manager-Kommuniktion und keine Möglichkeit umfangreiche Datenmengen austauschen zu können, führten 1993 zur Entwicklung von ***SNMPv2.*** [STA96]

Die grundlegende Kommunikation erfolgt in beiden Versionen zwischen Managern und Agenten über ein verbindungsloses, asynchrones Protokoll mit den PDU-Typen:

- *GetPDU:* Lesender Zugriff auf eine MIB-Variable des Agenten und deren Rückgabe.
- *GetNextPDU*: Lesender Zugriff auf eine MIB-Variable durch zeilenweises Durchwandern von Tabellenspalten und Rückgabe der Werte.
- *SetPDU*: Schreibender Zugriff auf eine MIB-Variable des Agenten.
- *TrapPDU*: Meldung durch den Agenten an den Manager ohne vorherige Aufforderung.
  Es existiert eine vordefinerte Menge an Trap-Meldungen (z.B.: coldStart, warmStart, linkDown, authenticationFailure, enterpriseSpecific ...)

3.3.4. MIB

(Internet-Management: Informationsmodell - SMI, **S**tructure of **M**anagement **I**nformation)

Zentrales Element des Internet-Informationsmodells ist die Management Information Base (MIB), die als Datenbehälter für managementrelevante Informationen dient. Die Informationen sind in einer Baumstruktur organisiert, dem Registrierungsbaum (im Gegensatz zum OSI-Management liegt hier weder ein Vererbungs- noch ein Enthaltenseinbaum vor). Jede Information stellt ein Managed Object (in ASN.1 beschrieben) dar, das sich ausschließlich in den Blättern befindet. [STA96]

3.3.5. MIB-II

Die MIB-II wird auch als Standard-MIB bezeichnet und kann von jedem Managementagenten interpretiert werden. Sie befindet sich im Registrierungsbaum unter iso(1)org(3)dod(6)internet(1)management(2)mib-2(1). [STA96]

Folgende Gruppen liegen unter anderem unter diesem Knoten:

- *system(1)*
  allgemeine Informationen über den Managed Node, Ansprechpartner, Systemname, Dienstebene...;
- *interfaces(2)*
  Anzahl und genauere Informationen (Typ, Statistiken, Konfiguration...) der Schnittstellen;
- *ip(4)*
- *tcp(6)*
- *udp(7)*
- *snmp(11)*
  Alle vier vorhergehenden Punkte gehören zu der „Protokoll"-Gruppe, die Informationen und Statistiken über Protokolleinheiten sowie Fehlersituationen enthalten.
- *transmission(10)*
  Informationen über Übertragungsprotokolle und Netzschnittstellen;

Nutzbare Topologie-Information:

- **IPForwarding (1.3.6.1.2.1.4.1.0)**
- **Systembeschreibung (1.3.6.1.2.1.1.1.0)**
- **Systemdienstebene (1.3.6.1.2.1.1.7.0)**
- **Anzahl der Interfaces (1.3.6.1.2.1.2.1)**
- **Interfaceadressen (1.3.6.1.2.1.2.2.1.6)**
- **Interfacetypen (1.3.6.1.2.1.2.2.1.3)**
- **Interfacebeschreibung (1.3.6.1.2.1.2.2.1.2)**
- **IP-Adressen (1.3.6.1.2.1.4.20.1.1)**
- **Subnetmask für jede IP (1.3.6.1.2.1.4.20.1.3)**
- **DNS-Name (1.3.6.1.2.1.1.5.0)**

3.3.6. DOT1Bridge-MIB

Die dot1Bridge-MIB liegt unterhalb der MIB-II und ist eine optionale Komponente. Innerhalb des Registrierungsbaumes befindet sie sich unter iso(1)org(3)dod(6)internet(1)management(2)mib-2(1)dot1Bridge(17). [STA96]

Nutzbare Topologie-Information:

- **Bridgeadresse (1.3.6.1.2.1.17.1.1.0)**
- **Anzahl der Ports (1.3.6.1.2.1.17.1.2.0)**
- **Liste aller MACAdressen (1.3.6.1.2.1.17.4.3.1.1)**
- **Zuordnung Port – MAC-Adresse (1.3.6.1.2.1.17.4.3.1.2)**

3.3.7. RMON

Sie liegt unterhalb der MIB-II und ist ebenfalls eine optionale Komponente. Innerhalb des Registrierungsbaumes befindet sie sich unter iso(1)org(3)dod(6)internet(1)management(2)mib-2(1)rmon(16).

Die Remote-Monitoring MIB beinhaltet Informationen zur genaueren Netzanalyse. Ihre Objekte stellen an den Agenten höhere Anforderungen, da sie bereits Auswertungen von

Netzstatistiken beinhaltet und durch diese Vorverarbeitung von Informationen den Netzverkehr verringert. [STA96]

Folgende Gruppen liegen unter anderem unter 1.3.6.1.2.1.16:

- *statistics(1)*
  Anzahl von Paketen, Bytes, Broadcast, Fehlertypen....;
- *history(2)*
  Trendanalyse, durch Aufzeichnung der Statistic-Group (aus MIB-II) möglich;
- *alarm(3)*
  Alarmauslösung (TRAPs) auf der Basis von Schwellwertanalysen;
- *host(4)*
  beinhaltet die HostTable, die für jeden Host bzw. für jede MAC-Karte den Netzverkehr analysiert;
- *event(19)*
  Ereignisdefinition, die bei Ereigniseintritt bestimmte Reaktionen auslöst;

Da es sich um eine optionale Komponente handelt, stand sie auf den analysierten Systemen leider nicht zur Verfügung.

3.3.8. PING

Ping (Packet Internetwork Groper) sendet an den Echoport (Portnummer 7) des innerhalb des Pingbefehls adressierten Endsystems UDP-Pakete und mißt die Zeit, bis das Echo zurückkommt. Dieses Programm kann verwendet werden um festzustellen, ob ein entferntes System am Netz angeschlossen und erreichbar ist.

Nutzbare Topologie-Information: System ist bis Ebene 3 (IP) funktionsfähig

3.4. Objektorientierte Modellierungstechnik

Im Kapitel „Realisierung“ wird verstärkt auf den Ablauf der Programme eingegangen. Um diesen zeitlich und informationsbedingt graphisch übersichtlich darstellen zu können, wird in diesem Abschnitt näher auf eine weit verbreitete Modellierungstechnik eingegangen. [MUE98]

3.4.1. UML

OMT, die Object Modelling Technique, eine objektorientierte Software-Engineering-Methode, basiert auf bekannten Entwurfstechniken wie dem Entity-Relationship-, Zustands- und Datenflußdiagramm. Aus ihr ist unter anderem UML [MUL97], die Unified Modelling Language, entstanden, die sich durch Verwendung leicht abgewandelter Diagramme unterscheidet. Diese objektorientierte Modellierungssprache wird von einer Vielzahl von CASE-Tools unterstützt, die für die Modellierung Diagrammeditoren zur Verfügung stellen und teilweise die graphisch entwickelten Klassen mit ihren Schnittstellen in unterschiedliche Programmiersprachen einbetten können. In dieser Arbeit wird RATIONAL-ROSE [RAT] verwendet, das CASE-Tool der Firma RATIONAL. (www.rational-software.de)

Aus UML werden folgende Diagramme für die Darstellung des Objektmodells verwendet:

- ***Klassendiagramm***

  Darstellung der Klassen und ihrer Relationen.

  Klassen beschreiben eine Menge von Objekten mit gemeinsamen Methoden (evtl. mit einer Argumentenliste „IN“ und einem Ergebniswert „OUT“) und Attributen (mit eindeutigem Datentyp). Klassen stehen in Beziehung zueinander, die durch folgende Relationstypen ausgedrückt werden

  - Assoziation (gleichberechtigte Partner);
  - Aggregation (Gesamtheit-Teil-Beziehung);
  - Vererbung (Superklasse-Subklasse Beziehung);

- **Zustandsdiagramm**

  Beschreibung des Verhaltens eines Objekte einer bestimmten Klasse während seiner Lebenszeit.

Der Zustand eines Objektes wird durch seine aktuellen Attributwerte und Beziehungen zu anderen Objekten beschrieben. Zustandsdiagramme sind gerichtete Graphen, die den Übergang eines Objektzustandes durch von außen eintreffende Ereignisse in einen Folgezustand darstellen. Diesen Übergang nennt man Transition.

- **Interaktionsdiagramm**

  Beschreiben die Zusammenarbeit verschiedener Objekte. Als Interaktion wird dabei der Nachrichtenaustausch zwischen den Objekten zur Erfüllung einer bestimmten Aufgabe verstanden.

- **Sequenzdiagramm**

  Darstellung der zeitlichen Reihenfolge, in der Nachrichten zwischen Objekten ausgetauscht werden.

4. Realisierung

4.1. Theorie und Praxis

Im folgenden soll nun kurz auf die Bearbeitungstaktik eingegangen werden.
Nach einer theoretischen Einarbeitungsphase in Rechnernetzgrundlagen sowie in die Verwaltung von Unix-Rechnernetzen stellte sich heraus, daß MIBs die informationsreichsten Datenlieferanten sind. Es bildete sich ein umfangreiches Gerüst von relevanten Variablen, das für die Auswertung von großem Interesse war. Vor allem [STA96] lieferte in aller Ausführlichkeit Informationen über MIB-Variablen.
Leider konnte diese theoretische Vorarbeit in der Praxis nicht viel weiterhelfen. Als die ersten SNMP-Abfragen entstanden, konnten aus der MIB-II, die zum Glück auf jedem System vorhanden ist, die wichtigsten Systeminformationen erfragt werden. Nur die optionalen Komponenten zeichneten sich durch Nichtvorhandensein aus und hinterließen in dem Informationsschema große Lücken.
So konnte sich einerseits aus Ermangelung implementierter MIBs und andererseits aus fehlenden Zugriffsrechten nur ein grobmaschiges Grundgerüst aus Informationen bilden.
Die zweite Hürde stellte das neue MASA-System dar. Ganz neu implementiert stand es ohne jegliche Dokumentation diesem FoPra zur Verfügung. So stellte sich die Importierung der laufenden Java-Programme in das System durch zusätzlich regelmäßig stattfindende Systemaktualisierung für einen „Nicht-Insider“ als keine allzu leichte Aufgabe dar.

Im folgenden wird der Agent aus Benutzersicht erklärt, indem die Maskeneingaben und die Ausgaben anhand von Beispielen dargestellt werden. Anschließend wird intensiver auf die Programmlogik eingegangen.

Das Agentenapplet

Die grundlegenden Werkzeuge für den Agenten sind wie bereits beschrieben DNS, NIS und SNMP. Zusätzlich müssen spezielle Daten vom Benutzer ergänzt werden. Dieser muß vor dem Start folgende Informationen zur Verfügung stellen (siehe Abbildung 1):

- Community-String: für die SNMP-Abfragen;
  - *verpflichtend*
- Filename: falls der Benutzer die Log-Informationen speichern will;
  - *optional*

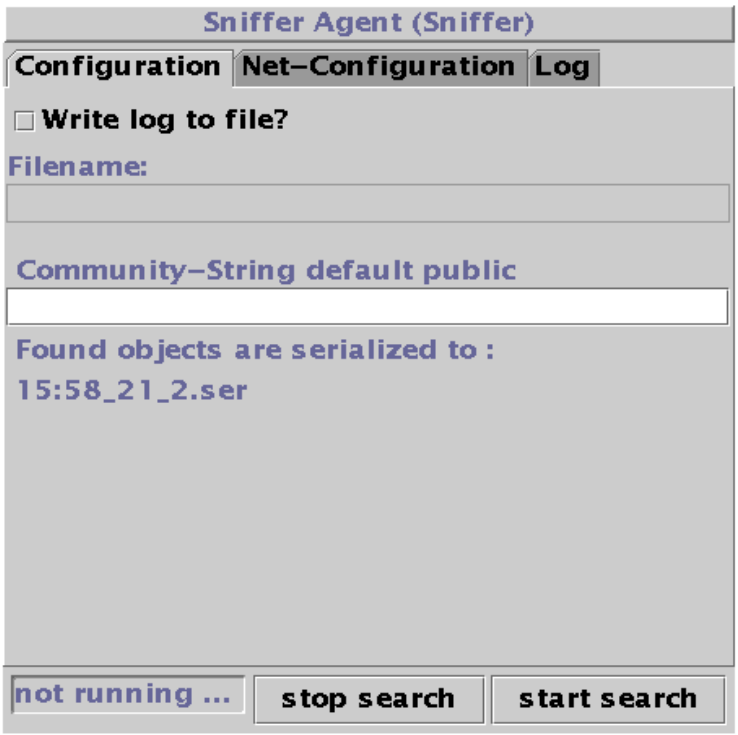

Abbildung 1: Benutzereingabe/Konfiguration

Die durch den Agenten gefundenen Ergebnisse werden automatisch serialisiert in ein File unter „/proj/fagent/tmp/SnifferResults/“ abgelegt, dessen Name sich aus aktueller Uhrzeit und Datum zusammensetzt („h:min_dd_mm.ser“).

Folgende Vektoren werden serialisiert (vgl. Abbildung 4):

1. Vektor: Liste aller nicht gefundenen NIS-Elementen;
2. Vektor: Liste aus gefunden Bridges;
3. Vektor: Liste aus MgmtElements (vgl. Abbildung 9);

- das zu untersuchende System (siehe Abbildung 2):
    - *verpflichtend;*

Dafür hat er vier Möglichkeiten:

1. Das gesamte lokale Netz: Der Agent verwendet hierfür als Anhaltspunkt die IP-Adresse des Hosts, auf dem er läuft.
2. Eine einzelne IP-Adresse;
3. Eine IP-Adress-Range, z.B. von 129.187.214.10 bis 129.187.214.20;
4. Ein fremdes komplettes Netz, dafür muß der Benutzer eine Netzadresse eingeben, z.B. 129.187.214.

Abbildung 2: Benutzereingabe: Netzangaben für Untersuchung

Sind diese Benutzerdaten eingegeben, kann der Agent gestartet werden.

In der Log-Maske (siehe Abbildung 3) kann man sich zur Laufzeit des Agent ansehen, welche Daten aktuell gefunden werden und die Ergebnisse des Agenten werden in ihr protokolliert (Diese Daten werden optional in einem Logfile gespeichert, falls es der Benutzer will, vgl. Abb1).
Da der Agent am Ende seines Programmdurchlaufes ist, wird in Abbildung 3 die Stop-Meldung angezeigt.

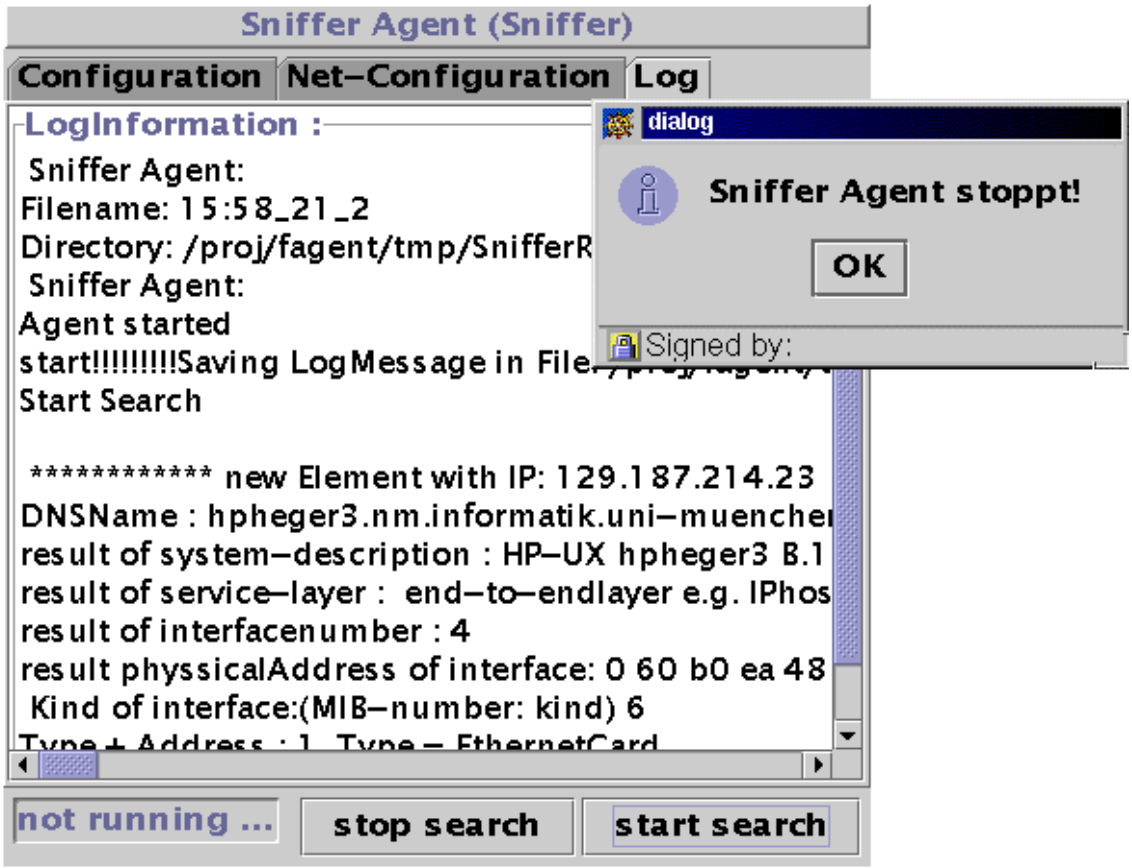

Abbildung 3 : Log-File incl. Ergebnisse

Zu sehen sind folgende Informationen:

- Die Ergebnisse werden im File 15:58_21_2.ser gespeichert, d.h. diese Ergebnisse wurden am 21.2. um 15.58 Uhr erziehlt.
- Dieses File liegt unter /proj/fagent/tmp/SnifferResults;
- Das untersuchte IP-Element hat die IP-Adresse 129.187.214.23
- DNS-Name....

Hier ist ein Beispiel eines gespeicherten Log-Files eines Testlaufes für die IP-Address-Range 129.187.214.23 – 129.187.214.26:

```
        !!!!!!!!!Saving LogMessage in File: /proj/fagent/tmp/SnifferResults/14:44_21_2
        Start Search

         ************ new Element with IP: 129.187.214.23
        DNSName : hpheger3.nm.informatik.uni-muenchen.de
        result of system-description : HP-UX hpheger3 B.10.20 A 9000/782 2003703982
        result of service-layer :  end-to-endlayer e.g. IPhost und application layer e.g. mail
        server
        result of interfacenumber : 4
        result physsicalAddress of interface: 0 60 b0 ea 48 83
         Kind of interface:(MIB-number: kind) 6
        Type + Address : 1. Type - EthernetCard
        ip-Addresse :  129.187.214.23
        Subnetmask :  255.255.255.0
        result of forwarding ( 1-true, 0-false): 1
        MgmtElement.Dns : hpheger3.nm.informatik.uni-muenchen.de
        MgmtElement.Descrption : HP-UX hpheger3 B.10.20 A 9000/782 2003703982
        MgmtElement.funkt : [4, 7]
        MgmtElement.numb_interf : 4
        MgmtElement.Forwarding : true
        MgmtElement.Element_nr : 1
        MgmtElement.vec_Interfaces :
        [de.unimuenchen.informatik.mnm.masa.agent.SnifferAgent.Interfaces@2e7aa4bf]
        MgmtElement.vec_IPElements :
        [de.unimuenchen.informatik.mnm.masa.agent.SnifferAgent.IPElement@5492a4bf]
        MgmtElement.Next_Bridges : []
⇒  !!  The IP did not answer, no new Element for IP: 129.187.214.24

         ************ new Element with IP: 129.187.214.25
        DNSName : hpheger5.nm.informatik.uni-muenchen.de
        result of system-description : HP-UX hpheger5 B.10.20 A 9000/750 2007422322
        result of service-layer :  end-to-endlayer e.g. IPhost und application layer e.g. mail
        server
        result of interfacenumber : 4
        result physsicalAddress of interface: 8 0 9 27 14 17
         Kind of interface:(MIB-number: kind) 6
        Type + Address : 1. Type - EthernetCard
        ip-Addresse :  129.187.214.25
        Subnetmask :  255.255.255.0
        result of forwarding ( 1-true, 0-false): 1
        MgmtElement.Dns : hpheger5.nm.informatik.uni-muenchen.de
        MgmtElement.Descrption : HP-UX hpheger5 B.10.20 A 9000/750 2007422322
        MgmtElement.funkt : [4, 7]
        MgmtElement.numb_interf : 4
        MgmtElement.Forwarding : true
        MgmtElement.Element_nr : 2
        MgmtElement.vec_Interfaces :
        [de.unimuenchen.informatik.mnm.masa.agent.SnifferAgent.Interfaces@4512a4b9]
        MgmtElement.vec_IPElements :
        [de.unimuenchen.informatik.mnm.masa.agent.SnifferAgent.IPElement@6a42a4b9]
        MgmtElement.Next_Bridges : []
⇒  !! The IP did not answer, no new Element for IP: 129.187.214.26
        Ende der Suche is_running: false
        !!!!!!!!!Saving LogMessage in File: /proj/fagent/tmp/SnifferResults/14:45_21_2
```

4.2. Der Programmablauf

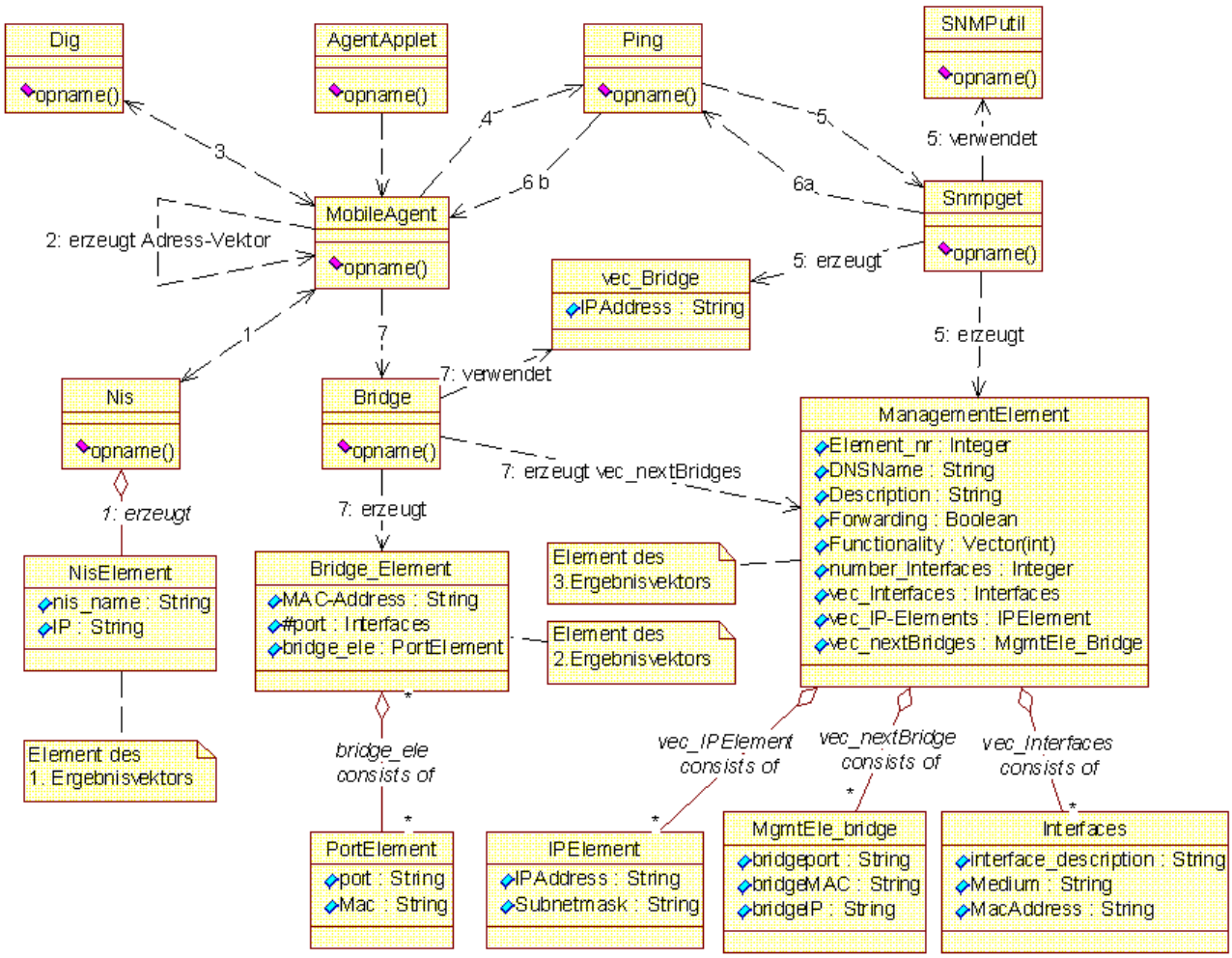

Abbildung 4: Sniffer-Architektur

4.2.1. Die Architektur

Abbildung 4 zeigt die Gesamtarchitektur des Sniffer-Agenten und die zeitliche Reihenfolge seiner Programmaufrufe. Es sind alle Klassen dargestellt, wobei lediglich die, die „opname“ enthalten, die Funktionalität des Agenten erfüllen.

Zu bemerken ist hier vielleicht noch, daß die zu Beginn mit dem Benutzer stattfindende Kommunikation lediglich im AgentApplet erfolgt.

4.2.2. SnifferAgentApplet

Liegen die benötigten Informationen IP-Adressen sowie Community-String durch die Benutzereingabe vor, kann der Agent gestartet werden.
Hier soll noch erwähnt werden, daß die komplette Kommunikation zwischen Applet und Agent über CORBA realisiert ist. Die verwendeten Schnittstellen müssen in der programmiersprachenunabhängigen IDL-Notation, der Interface-Definintion-Language, implementiert sein. Diese wird in einem idl-File festgelegt (siehe Anhang).

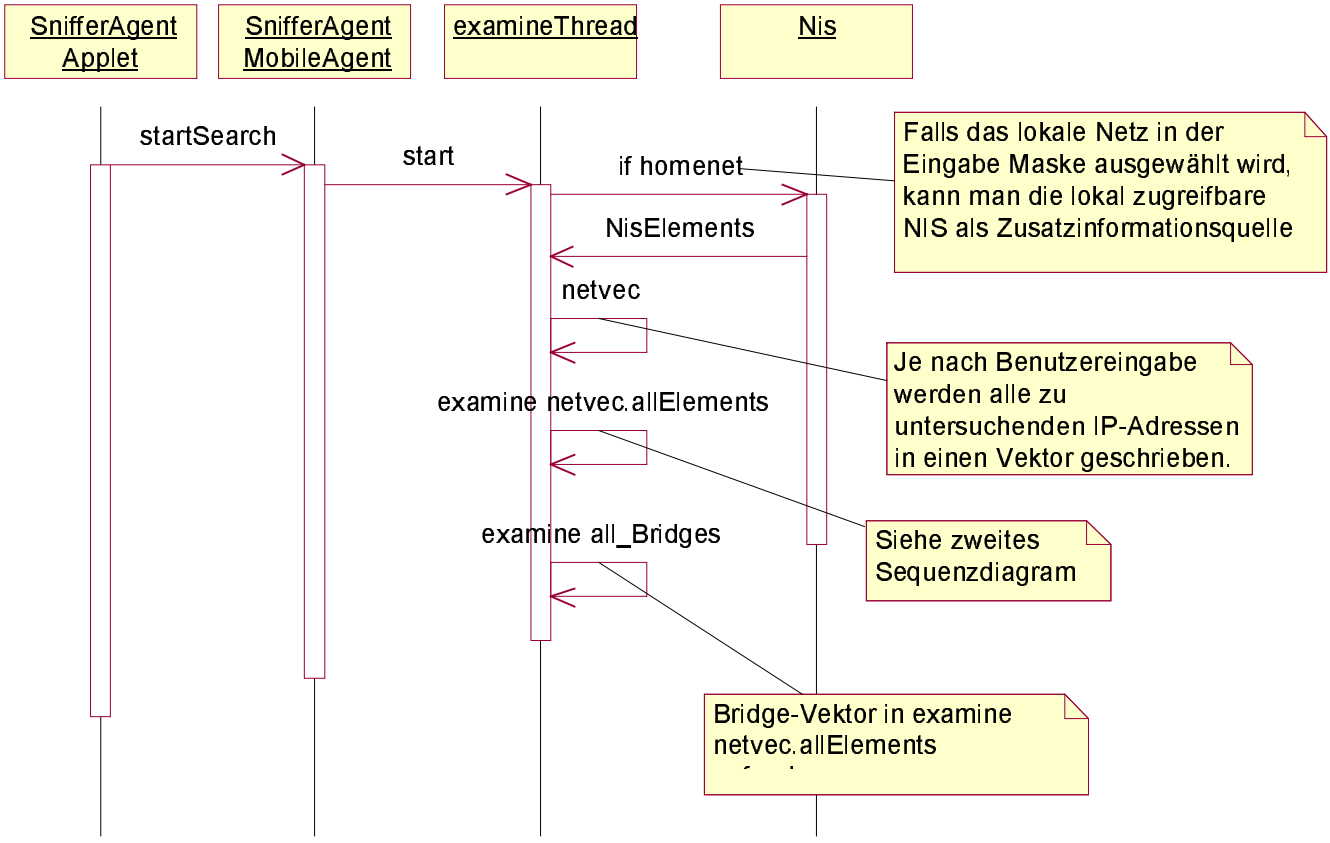

Abbildung 5: Sniffer Agent-Rahmenprogramm-Ablauf

4.2.3. SnifferAgentMobileAgent

Nach dem Start des Agenten wird in dem Hauptprogramm SnifferAgentMobileAgent zuerst der

- **examineThread**

aufgerufen, der den Vektor aus zu untersuchenden IP-Adressen erzeugt.
Handelt es sich um das gesamte lokale Netz, so wird zusätzlich durch

- **NIS**

die NIS-Host-Datei ausgelesen und zur Kontrolle alle gefundenen NisElemente (siehe Abbildung 6) als Vektor gespeichert.

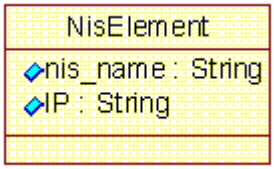

Abbildung 6: NIS-Element

Später wird jedes IP-Element, das während der Agentensuche gefunden wird, aus dem NIS-Vektor gelöscht. Am Ende des Durchlaufes werden die übriggebliebenen NIS-Elemente als „Vergleichskontrolle“ (welche existenten Hosts wurden nicht gefunden) mit in das serialisierte File geschrieben.

4.2.4. examine netvec.allElements

In diesem Programm wird jede IP-Adresse aus dem zu Beginn erzeugten Vektor schleifenartig durchlaufen.

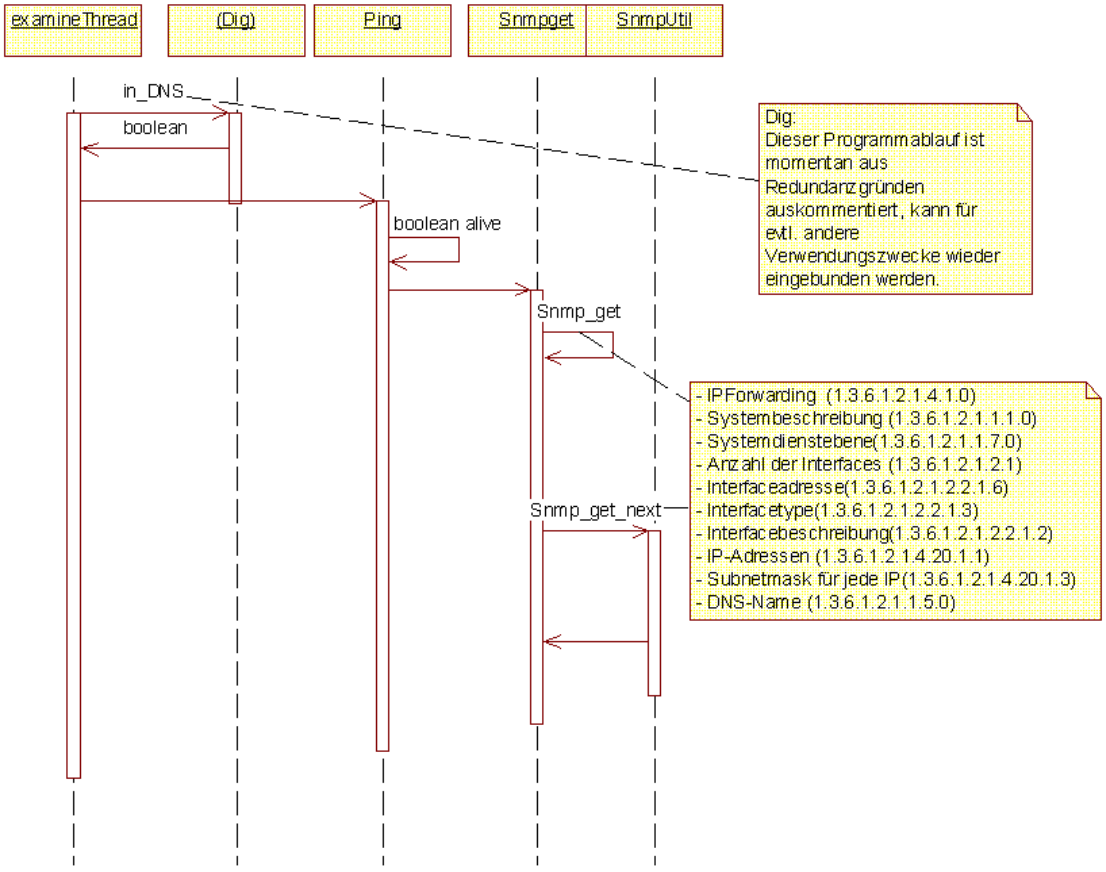

Abbildung 7: IP-Adressen Analyse

Für jede IP-Adresse findet folgende Überprüfung statt:

- **DNS:**

Diese Abfrage war zu Beginn der Programmentstehung als Zusatzinformation interessant. Da der DNS-Name aber einerseits aus der MIB-II ausgelesen werden kann und andererseits DNS auch keine verläßliche Informationsquelle über die tatsächliche Existenz der IP-Adressen gibt, wurde diese Abfrage aus dem Ablauf herausgenommen. (Der Vollständigkeit zu liebe ist die DNS-Abfrage noch in Abbildung 6 integriert und das Programm kann durch einfaches Entfernen eines Kommentarzeichens wieder integriert werden.)

- **PING:**

Hier wird gestestet, ob das System mit der gegebenen IP-Adresse wirklich im Netz vorhanden ist bzw. ob das System ansprechbar ist.

4.2.5. Snmpget

Dieses Programm liefert die Ergebnisse zu folgenden MIB-Variablen:

1. **IPForwarding:**

   1 = true (d.h. dieses System arbeitet auf Ebene 4 und „forwarded“ Pakete)

   0 = false;

2. **Systemdescription**

   Eine textuelle Beschreibung, kann evtl. Zusatzinformationen geben;

3. **Anzahl der Interfaces**

   Integerwert der alle Interfaces (nicht nur die physischen Netzkarten) widerspiegelt;

4. **Interfaceadressen**

   alle MAC-Adressen;

5. **Interfacetypen**

   1 = others;

   6 = ethernet Csmacd protocol;

   7 = iso88023 Csmacd IEEE802.3 CSMA/CD MAC protocol;

   8 = iso88024 TokenBus IEEE802.4 Token Bus MAC protocol;

   9 = iso88025 TokenRing IEEE802.5 Token Ring MAC protocol;

   11 = starLAN Mbps TwistedPair version for Ethernet;

   15 = fddi : ANSI Fiber Distributed Data Interface (FDDI) standard LAN ;

6. **Interfacebeschreibungen**

   Interfacebeschreibung in Prosa;

Da jedes System mehrere MAC-Interfaces besitzen kann, die jede für sich eine eigene Adresse, einen eigenen Typ und eine eigene Beschreibung haben kann, wird aus den letzten vier Informationen wieder ein Vektor zusammengesetzt (siehe Abbildung 9).
Bei diesen Ergebnissen ist wieder ein besonderes Augenmerk auf den Interfacetyp zu werfen, da dieser Informationen über die Topologie enthält.

7. **IP-Adressen**
8. **Subnetmask für jede IP**

   Genau wie bei den MAC-Adressen kann jedes System mehrere IP-Adressen mit zugehöriger Subnetmask besitzen, so daß ein IP-Vektor angelegt wird (siehe Abbildung 9).

9. **DNS-Name**
10. **Systemdienstebene**

    1 = physical layer z.B. Repeater,
    2 = datalinklayer z.B. Bridge,
    3 = internet layer z.B. IProuter,
    4 = end-to-endlayer z.B. IPhost,
    7 = application layer z.B. Mail Server,
    6 = (Schicht 2&3) datalinklayer z.B. Bridge und internet layer e.g. IPRouter,
    12 = (Schicht 3&4) end-to-endlayer z.B. IPhost und internet layer z.B. IPRouter,
    72 = (Schicht 4&7) end-to-endlayer z.B. IPhost und application layer z.B. Mail Server;

Aus dem Ergebnis der Dienstebene lassen sich die einzigen Rückschlüsse auf die Funktionsweise des Systems ziehen. Hier entscheidet es sich, ob es sich um ein Koppelelement oder ein Endsystem handelt.

4.2.6. Das Bridge-Modul

Für „Bridges“ wurde eine eigene Routine geschrieben, die genauer auf die Topologie-Erforschung eingeht. Praktisch konnte diese jedoch leider nicht getestet werden, da nur eine Bridge zur Verfügung stand, die durch netzfremde IP-Adressen und fremden Community-String nicht in die automatische Suche mit aufgenommen werden konnte.

Theorie:

Wird durch die SNMP-Serviceebene eine Brücke identifiziert, so wird ihre IP-Adresse in den Vektor „vec_Bridge“ eingetragen.

Dieser Vektor wird (wie in Abbildung 5 ersichtlich) nach vollständiger Abarbeitung des „allElements“-Vektors elementweise durchlaufen, d.h. für jede IP-Adresse wird das Bridge-Snmp-Programm aufgerufen, das folgende Variablen analysiert (vgl. 3.4.6) :

- Bridgeadresse
- Anzahl der Ports
- Liste aller MACAdressen
- Zuordnung Port – MAC-Adresse

Ein vollständiges Bridge_Element sieht wie folgt aus:

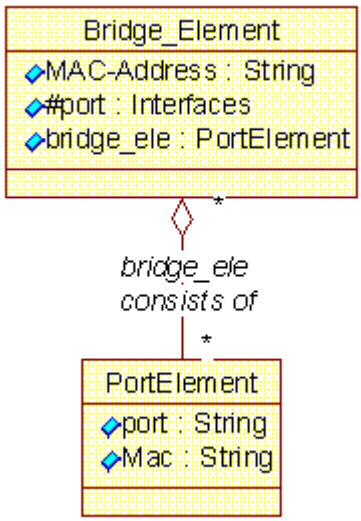

Abbildung 8 :Bridge_Element

Die Variable Port-MAC-Adresse ist das interessanteste Ergebnis. Für jede hier gefundene MAC-Adresse wird die Liste der MgmtElements durchlaufen, d.h. es wird untersucht, ob eines der bereits gefundene MgmtElements aktiv hinter dieser Brückenport hängt. Durch diese Information ist es möglich die topologische Beziehung Bridge – Endsystem aufzubauen. Wird also in dem „allElement“-Vektor diese MAC-Adresse gefunden, so wird in die Variable MgmtElement.Bridge diese Bridge-Adresse hineingeschrieben. Am Ende der Untersuchung kann jedes MgmtElement Adressen in seinem BridgeVektor haben, anhand derer genau festgestellt werden kann, zwischen bzw. hinter welchen Koppelementen sich die Endsysteme befinden.

4.2.7. Finale

Sind nun alle IP-Adressen aus dem erstellten Startvektor und der gesamte Bridge-Vektor abgearbeitet, so werden die Ergebnisse abgespeichert.
Hier nocheinmal die graphische Darstellung des ManagementElements, das die erzielten Ergebnisse beinhaltet.

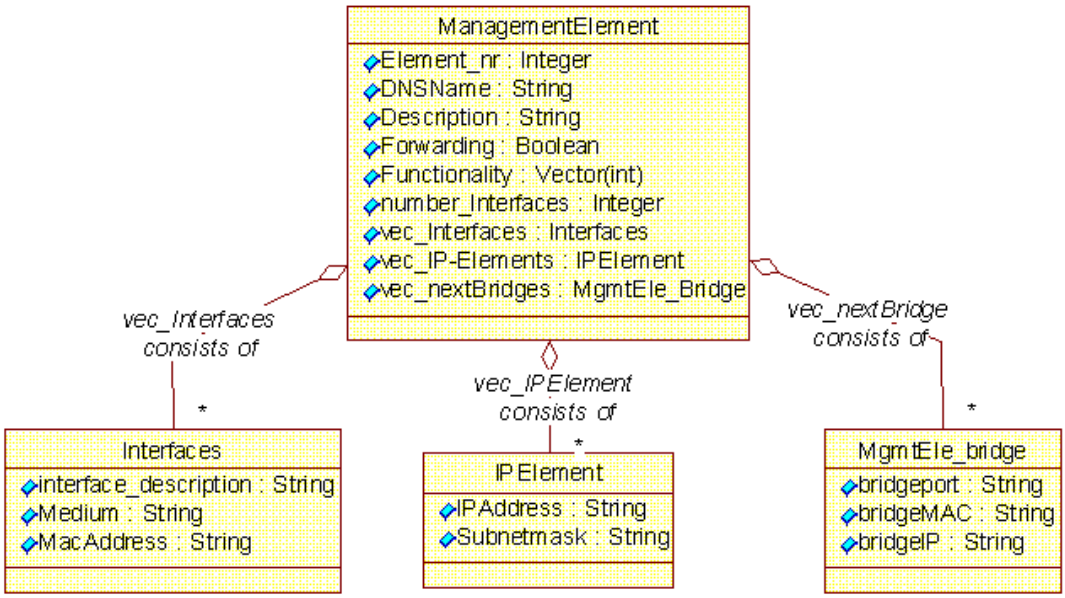

Abbildung 9: ManagementElement

5. Ausblick und Zusammenfassung

Dieses Fortgeschrittenenpraktikum hat eine Auswahl von Topologiedaten zusammengetragen und diese ansatzweise ausgewertet. Diese Ergebnisse könnten als Input für weitere Projekte wie zum Beispiel einer graphischen Darstellung, Hardware-Analysen / Hardware-Inventur oder anderen Hardwareinformationen benötigenden Diensten verwendet werden. Da der Agent selbst bereits eine ideale Schnittstelle zu den Endgeräten darstellt, könnte seine Funktionalität beliebig erweitert werden. D.h. es könnten sehr leicht weitere Informationen z.B. aus MIBs für Accounting-Zwecke erhalten oder weitere Hardwareabfragen der Systeme getätigt werden. Auch eine Erweiterung um Benutzerzugriffe zu überwachen wären möglich.
Attraktiv wäre desweiteren, die existierende Funktionalität zu erweitern, indem man z.B. Kombinationen von Netzwerk-, Range- oder Single-IP-Adressen ermöglicht oder die Nis als Startinformation verwendet. Die Verwendung von mehreren Communitiystring wäre für die vollständige Netzanalyse sehr hilfreich, ist jedoch mit großer Vorsicht zu genießen, da durch die dopplten/dreifachen oder vierfachen Abfagen die Netzwerklast stark erhöht werden kann.

6. Anhang

genPropertiesFile.SnifferAgent

```
#!/bin/sh
#
# Generate properties of the agent.
# These properties are automatically read by the agent system
# once the agent is created.
#

if [ ! ${MAKELEVEL} ]; then
  echo "Please call $0 from the Makefile!"
  exit 1
fi

echo "
##############################################################################################
#
# DO NOT EDIT. Generated by ${0}.
#
##############################################################################################

##############################################################################################
#
#  Properties needed by the SnifferAgent agent
#

${MASA_PACKAGE_thisagent}.dig0=/soft/bin/dig
${MASA_PACKAGE_thisagent}.dig1= -x
${MASA_PACKAGE_thisagent}.nis0=/usr/bin/ypcat
${MASA_PACKAGE_thisagent}.nis1= hosts
${MASA_PACKAGE_thisagent}.save=/proj/fagent/tmp/SnifferResults/


"
```

copyForeignClassesToJar

```
#!/bin/sh -x
#usage: copyForeignClassToJar <package_path>

while [ ! "$#" = "0" ]; do
  PACKPATH=$1
  cd ${PACKPATH}  ; ${FIND_CMD}  -name "*.class" -print > ${BUILD_TMP_PATH}/installhelper.tmp
    rm -r  ${ROOT_PATH}/${PROD_DIR}/Snmp
  mkdir ${SNIFFER_RESULTS}
  mkdir ${ROOT_PATH}/${PROD_DIR}/Snmp
 ${TAR_CMD}   -c -T ${BUILD_TMP_PATH}/installhelper.tmp | ${TAR_CMD} -xv -C
${ROOT_PATH}/${PROD_DIR}/Snmp
  shift 1
done
```

genPolicyWishList.SnifferAgent

```
#!/bin/bash
#
# Shell script to write a wishlist.policy file to stdout
# This file is read by the agent system to determine which basic needs
# an agent has.
#

if [ ! ${MAKELEVEL} ]; then
  echo "Please call $0 from the Makefile!"
  exit 1
fi


echo "
//############################################################################################
```

```
//
// DO NOT EDIT. Generated from ${0}.
//
//#############################################################################################

//#############################################################################################
//
// Special permissions needed by the SnifferAgent agent
//
permission java.io.FilePermission \"/soft/bin/dig\", \"read, execute\";
permission java.io.FilePermission \"/usr/bin/ypcat\", \"read, execute\";

//permission java.io.FilePermission \"<<ALL FILES>>\", \"write, read\";

permission java.io.FilePermission \"/proj/fagent/tmp/SnifferResults/*\", \"write\";

permission java.net.SocketPermission \"*.informatik.uni-muenchen.de\",
\"accept,connect,resolve,listen\";
permission java.net.SocketPermission \"141.84.220.250\", \"accept,connect,resolve,listen\";

permission ${MASA_PACKAGE_agentSystem}.ClassAllowUsePermission
\"de.unimuenchen.informatik.mnm.masa.agent.SnifferAgent.classes.Snmp.*\";
permission ${MASA_PACKAGE_agentSystem}.ClassAllowUsePermission \"Snmp.*\";
permission ${MASA_PACKAGE_agentSystem}.ClassAllowDefinePermission \"Snmp.*\";

"
```

SnifferAgent.idl

```
// CORBA IDL for agent SnifferAgent

// the next two lines are C preprocessor directives
// which are allowed in CORBA IDL
#ifndef _SnifferAgent_idl_
#define _SnifferAgent_idl_

// this will be a mobile agent so we need Migration.idl
#include "Migration.idl"

// this module will be converted to a java package of the agent
module SnifferAgent {

  // the interface SnifferAgent has to extend 'Migration'
  // which is inside the module 'agent'
  interface SnifferAgent : agent::Migration
  {
     // here go the operation offered by this agent


      string getLogInformation();
      boolean isRunning();
      string getSaveName();

      void setCommunityString(in string CommunityString);
      void setFileName(in string FileName);
      void setWriteToFile(in boolean WriteToFile);
      void setAddress(in string GetAddress);
      void setNetAddress(in string GetNetAddress);
      void setAddressRangeTo(in string GetAddressRangeTo);
      void setAddressRangeFrom(in string GetAddressRangeFrom);
      void setSingleAddress(in string GetSingleAddress);
      void startSearch();
      void stopSearch();
      void setValue(in boolean Running);
      void resetLog();
  };
};

#endif /* of _SnifferAgent_idl_ */
```

7. Abbildungsverzeichnis

Abbildung 1: Benutzereingabe/Konfiguration 18
Abbildung 2: Benutzereingabe: Netzangaben für Untersuchung 19
Abbildung 3 : Log-File incl. Ergebnisse 20
Abbildung 4: Sniffer-Architektur 22
Abbildung 5: Sniffer Agent-Rahmenprogramm-Ablauf 23
Abbildung 6: NIS-Element 24
Abbildung 7: IP-Adressen Analyse 25
Abbildung 8 :Bridge_Element 28
Abbildung 9: ManagementElement 29

8. Literaturverzeichnis

FLA98 Flanagan, D.: Java in a Nutshell, 2. Auflage, O´Reilly 1998

GOK98 Golias, D., Kacalek, M.: Simple Network Management Protokoll Version 3
(SNMPv3), LMU München, Mai 1998

HUG99 Hughes, M., Shoffner, M., Hamner, D.: Java Network Programming, Manning 1999

KER93 Kerner, H.: Rechnernetze nach OSI, 2. Auflage, Addison-Wesley, 1993

KEM97 Kempter, B.: Realisierung eines Managementwerkzeugs in Java für das Management von IP-Netzen, LMU München, November 1997

KEM98 Kempter, B.: Entwurf eines Java/CORBA-basierten Mobilen Agenten, LMU München, August 1998

MAU99 Maul, O.: Einführung in Verzeichnisdienst am Beispiel DNS, LMU München, Mai 1999

MIT99 Mikulasch, P., Thaens, K.: Objektorientierte Software Entwicklung; Skript zur gleichnamigen Vorlesung im WS 1998/1999, LMU München, März 1999

MUE98 Müller, T.: CORBA-basiertes Management von UNIX-Workstations mit Hilfe von ODP-Konzepten, LMU München, Februar 1998

MUL97 Muller, P.: Instant UML; WROX Press, Birmingham 1997

RAT Rational Rose www.rational.com

ROE99 Rölle, H.: Realisierung des Policymanagements in MASA, LMU München, August 1999

STA96 Stallings, W.: SNMP, SNMPv2 and RMON, 2. Edition, Addison-Wesley, 1996

STE93 Stern, Hal.: NFS und NIS, Managing von Unix-Netzwerken, 1.Auflage, Addison-Wesley, 1993

TAN97 Tannenenbaum, A.: Computer-Netzwerke, 3. Auflage, Prentice Hall Verlag GmbH, 1997