DIPLOMARBEIT

Ankopplung des Feldbussystems P-NET an das Internet via SNMP

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-Ingenieurs
am Institut für Computertechnik der Technischen Universität Wien

unter der Anleitung von
o. Univ. Prof. Dipl.-Ing. Dr. techn. Dietmar Dietrich

und
den Betreuern

Dipl.-Ing. Dr. techn. Mag. Martin Manninger
Dipl.-Ing. Martin Knizak
Dipl.-Ing. Mikhail Gordeev

durch

Melih AYAL

Pfeilgasse 3a
A-1080 Wien
Matr.Nr. 9126504

Wien, 10.08.1999

_____
(Unterschrift)

KURZFASSUNG

In den letzten Jahren ist die Verbreitung von Feldbussen deutlich gestiegen. Durch einen direkten Zugriff auf ein Feldbussystem aus einem übergeordneten Netz, wie Internet, können Daten über größere Entfernungen hinweg erfaßt und verarbeitet werden. Diese Ankopplung des Feldbussystems ans Internet bietet mehrere Möglichkeiten, wie zum Beispiel zentrale Betriebsdatenerfassung, Fernüberwachung oder Fernwartung. Dieser Vernetzung soll an Hand eines weit verbreiteten und leicht durchschaubaren Protokolls erfolgen, damit kein detailliertes Fachwissen dafür erforderlich ist. Ein solches Protokoll ist das Simple Network Management Protocol (SNMP).

Da es momentan in der Industrie eine große Vielfalt von Feldbussystemen zu finden gibt, ist es sinnvoll, daß ein System konzipiert wird, welches so flexibel und modular aufgebaut ist, daß die Anbindung unterschiedlicher Feldbusse an das Internet mit geringstem Aufwand ermöglicht. Ein solches System soll aus einem protokollunabhängigen und einem feldbusspezifischen Teil bestehen. Durch diesen modularen Aufbau beschränkt sich die Arbeit für das Hinzufügen anderer Feldbussysteme beim Vorhandensein des allgemeinen Teils auf die Implementierung eines entsprechenden Feldbus-Teils.

In der vorliegenden Ausarbeitung werden zuerst die theoretischen Grundlagen dargestellt und aufbauend auf diesen wird eine Möglichkeit der Anbindung verschiedener Feldbussysteme an das Internet via SNMP vorgestellt. Letztendlich wird die Realisierung der Ankopplung des Feldbussystems P-NET ans Internet als ein Beispiel für das oben genannte System erläutert.

ABSTRACT

In the last years the distribution of fieldbusystems rose enormously. By direct access to a fieldbus system from a superordinate network, like Internet, data can be gathered and processed over larger distances. This connection of the fieldbussystem to the internet offers several possibilities, for example central industrial data capture, remote supervision or remote maintenance. This networking is to take place on the basis of a wide-spread and easily transparent protocol, so that no detailed specialized knowledge is necessary for it. Such a protocol is the Simple Network Management Protocol (SNMP).

Since there is a large variety of fieldbussystems to find in the industry at the moment, it is advisable that a system is conceived which is so flexible and modularly structured. Thus, the connection of different fieldbusses to the Internet would enable the smallest possible expenditure. Such a system is to consist of a protocol-independent and a fieldbus-specific section. By this modular structure the work of adding other fieldbussystems is limited to the implementation of an appropriate fieldbus section.

In this work firstly, the theoretical bases will be represented and constructed. Secondly, a possibility of the connection of different fieldbussystems to the Internet via SNMP will be shown. Finally the implementation of the connection of the fieldbus P-NET to the Internet is described as an example for the system specified above.

.

DANKSAGUNG

Wenn man nach vielen Jahren endlich den Abschluß einer Ausbildung erlangt hat, so ist es an der Zeit all jenen zu danken, die zur Erlangung derselben beigetragen haben. Darum will ich hier an erster Stelle meinen Eltern danken, die mich in den vielen Jahren immer unterstützt haben.

Außerdem möchte ich allen Leuten danken, die bei der Erstellung dieser Diplomarbeit geholfen haben. Mein besonderer Dank gilt meinen Betreuern o.Univ.Prof. Dipl.-Ing. Dr. techn. Dietmar Dietrich, Dipl.-Ing. Dr.techn. Thilo Sauter, Dipl.-Ing. Dr. techn. Mag. Martin Manninger, Dipl.-Ing. Martin Knizak, Dipl.-Ing. Michael Kunes und Dipl.-Ing. Mikhail Gordeev für ihre Vorschläge und tatkräftige Unterstützung.

Weiters möchte ich mich ganz herzlich bei meiner Freundin, Fr. Nergiz Eyüpoglu, für ihre Korrekturarbeiten an dieser Arbeit bedanken.

INHALTSVERZEICHNIS

ABKÜRZUNGEN 7

1. NETZWERKPROTOKOLLE 8
- 1.1 Das ISO/OSI-Modell 8
- 1.2 Die Internet-Protokollfamilie 10
  - 1.2.1 IP 10
  - 1.2.2 TCP 11
  - 1.2.3 UDP 11

2. SNMP UND NETZMANAGEMENT 12
- 2.1 Einführung in das Netzmanagement 12
- 2.2 Komponente des Netzmanagements 12
  - 2.2.1 Verwaltete Knoten 12
  - 2.2.2 Netzmanagement-Station 13
  - 2.2.3 Netzmanagement-Protokoll 14
  - 2.2.4 Netzmanagementinformation 15
- 2.3 SNMP 15
- 2.4 SNMP im OSI-Modell 16
- 2.5 Structure of Management Information 16
- 2.6 Management Information Base 23
- 2.7 Operationen von SNMPv1 28
- 2.8 SNMP Protocol Data Unit (PDU) 30
- 2.9 SNMPv2 32
- 2.10 SNMPv3 33

3. DAS FELDBUSSYSTEM P-NET 35
- 3.1 Die Grundprinzipien von P-NET 35
- 3.2 Die Architektur von P-NET 37
  - 3.2.1 Die Bitübertragungsschicht 37
  - 3.2.2 Die Sicherungsschicht 37
  - 3.2.3 Die Vermittlungsschicht 38
  - 3.2.4 Die Transportschicht 40
  - 3.2.5 Die Anwendungsschicht 40
- 3.3 Die Channel-Struktur 40
- 3.4 Process-Pascal 41
- 3.5 VIGO 43
  - 3.5.1 VIGOSERV 44
  - 3.5.2 Manager Information Base (MIB) 46
  - 3.5.3 Instruction Data Converter (IDC) 47
  - 3.5.4 HUGO2 48
  - 3.5.5 Network Drivers 48

4. EINBETTUNG VON FELDBUSSYSTEMEN INS NETZMANAGEMENT 49
- 4.1 Einführung 49
- 4.2 Kopplungsmöglichkeiten zwischen LAN und Feldbus 49
- 4.3 Struktur des Gateways 52
- 4.4 Master-Agent 54
- 4.5 Sub-Agent 56
- 4.6 Verbindung zwischen dem Master- und Sub-Agent 59

4.7 Kommunikationsprotokoll zwischen dem Master- und Sub-Agent 61
5. KONFIGURATOR VON SUB-AGENT FÜR P-NET 70
5.1 Allgemeine Eigenschaften des Konfigurators....70
5.2 Initialisierungsphase 70
5.3 Benutzeroberfläche des Konfigurators 71
5.4 Auswählen eines Objekts 74
6. SUB-AGENT FÜR P-NET....77
6.1 Allgemeine Eigenschaften 77
6.2 Initialisierungsphase des Sub-Agents 78
6.3 Multithreading im Sub-Agent 82
6.4 Kommunikation mit Master-Agents 85
6.5 Testprogramm für den Sub-Agent....86
7. RESÜMEE....89
LITERATURVERZEICHNIS 91
ABBILDUNGSVERZEICHNIS....94

ABKÜRZUNGEN

ASI	AKTUATOR-SENSOR-INTERFACE
ASN.1	ABSTRACT SYNTAX NOTATION ONE
CCITT	COMITÉ CONSULTATIF INTERNATIONAL TÉLÉGRAPHIQUE ET TÉLÉPHONIQUE
CMIP	COMMON MANAGEMENT INFORMATION PROTOCOL
CMOT	CMIP OVER TCP/IP
CMOL	CMIP OVER LOGICAL LINK CONTROL
DLL	DYNAMIC LINK LIBRARY
DNS	DOMAIN NAME SERVICE
DPI	DISTRIBUTED PROGRAM INTERFACE
EIA	ELECTRONIC INDUSTRIES ASSOCIATION
FIFO	FIRST IN FIRST OUT
IAB	INTERNET ACTIVITIES BOARD
IDC	INSTRUCTION DATA CONVERTER
IEC	INTERNATIONAL ELECTROTECHNICAL COMMITEE
IEEE	INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS
ISO	INTERNATIONAL ORGANIZATION OF STANDARDIZATION
LAN	LOCAL AREA NETWORK
MIB	MANAGEMENT INFORMATION BASE
MMS	MANUFACTURING MESSAGE SPECIFICATION
MPD	MESSAGE PROCESSING AND DISPATCHING
NMS	NETWORK MANAGEMENT STATION
NOC	NETWORK OPERATIONS CENTER
OID	OBJECT IDENTIFIER
OLE	OBJECT LINKING & EMBEDDING
OCX	OLE CONTROL EXTENSION
OSI	OPEN SYSTEM INTERCONNECTION
PDU	PROTOCOL DATA UNIT
RMON	REMOTE NETWORK MONITORING
SAR	SUB-AGENT REGISTRATION
SGMP	SIMPLE GATEWAY MONITORING PROTOCOL
SMI	STRUCTURE OF MANAGEMENT INFORMATION
SMUX	SNMP MULTIPLEXING
SNMP	SIMPLE NETWORK MANAGEMENT PROTOCOL
SWNO	SOFTWIRE NUMBER
USM	USER SECURITY MODEL
VBAC	VIEW-BASED ACCESS CONTROL
VIGO	VIRTUAL INTERFACE USING GLOBAL OBJECTS
WAN	WIDE AREA NETWORK

1. NETZWERKPROTOKOLLE

1.1 Das ISO/OSI-Modell

Für die Kommunikation offener Systeme erstellte die International Standardization Organization (ISO) 1980 ein Refenzmodell, nämlich das Open-Systems-Interconnection-Referenzmodell (OSI-Referenzmodell). Ein offenes System ist nach ISO ein solches, das nach außen mit genormten Protokollen kommuniziert, so daß die Kommunikation mit jedem beliebigen Partner, der sich auch an diese Protokolle hält, möglich ist.

Das OSI-Referenzmodell ist ein Hilfsmittel für Entwickler, das keine verbindlichen Aussagen über die Hardware- und Softwaregestaltung macht, sondern nur den Rahmen für die Entwicklung von Kommunikationsschnittstellen und Protokollen bestimmt. Es stellt auch eine Basis für die Einordnung der bereits existierenden Normen dar.

Das OSI-Modell ist in sieben Schichten unterteilt, während jede Schicht eine Funktion hat, die wieder aus bestimmten Subfunktionen besteht (siehe Abb. 1.1). Jede Schicht bietet der darüberliegenden ihre Dienste an und nimmt die der darunterliegenden in Anspruch. Die Verbindung zwischen zwei benachbarten Schichten wird als Schnittstelle bezeichnet. Die untersten vier OSI-Schichten stellen Ende-zu-Ende-Dienste dar, die für die Übertragung von Daten zuständig sind. Sie werden auch als Transportprotokolle bezeichnet. Die oberen drei Schichten stellen Anwendungsdienste zur Verfügung, die für die Weitergabe von Informationen verantwortlich sind. Deswegen nennt man sie Anwendungsprotokolle.

7. Anwendungsschicht (Application Layer)	**Anwendungs-protokolle**
6. Darstellungsschicht (Presentation Layer)	
5. Kommunikationssteuerungschicht (Session Layer)	
4. Transportschicht (Transport Layer)	**Transport-protokolle**
3. Vermittlungsschicht (Network Layer)	
2. Sicherungsschicht (Link Layer)	
1. Bitübertragungsschicht (Physical Layer)	

Abb. 1.1 Schichten des OSI-Referenzmodells

Im OSI-Referenzmodell gibt es folgende Schichten:

- Schicht 1: Bitübertragungsschicht (Physical Layer)

Diese Schicht übernimmt die Steuerung des physikalischen Übertragungsmediums innerhalb des Kommunikationssystems. Sie ist verantwortlich für die ungesicherte Übertragung von Bitströmen in korrekter Reihenfolge. In dieser Schicht wird auch festgelegt, wie eine physische Verbindung aktiviert bzw. deaktiviert werden soll.

- Schicht 2: Sicherungsschicht (Link Layer)

  Laut ISO-Standard ist die Aufgabe dieser Schicht das Sichern der Übertragung auf den einzelnen Teilstrecken des gesamten Übertragungsweges. In dieser Schicht werden Fehler erkannt und möglichst behoben.

- Schicht 3: Vermittlungsschicht (Network Layer)

  Nach ISO-OSI-Bestimmungen hat diese Schicht als Aufgabe die Vermittlung und den Aufbau des gesamten physikalischen Übertragungsweges (Routing).

- Schicht 4: Transportschicht (Transport Layer)

  Die Aufgabe dieser Schicht ist laut ISO-OSI-Bestimmungen das Festlegen der Transportverbindungen (=logische Verbindungen) von der Nachrichtenquelle zur Nachrichtensenke. In dieser Schicht werden diese Verbindungen auch überwacht, so daß die Qualität der Übertragung sichergestellt ist.

- Schicht 5: Kommunikationssteuerungsschicht (Session Layer)

  In dieser Schicht wird festgelegt, wie die Kommunikation zwischen zwei Partnern in einer Sitzung erfolgen soll, so daß ein geregelter Kommunikationsablauf gewährleistet wird. Diese Schicht erlaubt das Hinzufügen von Kontrollmechanismen beim Datenaustausch. Zu den Aufgaben gehört auch das Synchronisieren der Sitzungsparameter.

- Schicht 6: Darstellungsschicht (Presentation Layer)

  Die Darstellungsschicht stellt sicher, daß die Kommunikationspartner miteinander kommunizieren können, auch wenn sie unterschiedliche Datenstrukturen für ihre Pakete und Meldungen benutzen. Sie übersetzt die Daten der Partner in die Syntax des benutzten Kommunikationssystems, welche auch in dieser Schicht festgelegt wurde.

- Schicht 7: Anwendungsschicht (Application Layer)

  Laut ISO enthält diese Schicht die Dienstelemente zur Unterstützung von Anwendungsprozessen. Sie ist verantwortlich für die Aufrechterhaltung der Verbindung zwischen den Anwendungen.

1.2 Die Internet-Protokollfamilie

Die Internet-Protokollfamilie verwendet wie das OSI-Modell eine Schichtarchitektur (siehe Abb. 1.2). Sie besitzt nicht so viele Schichten wie das OSI-Modell und ist viel einfacher strukturiert und somit leichter implementierbar.

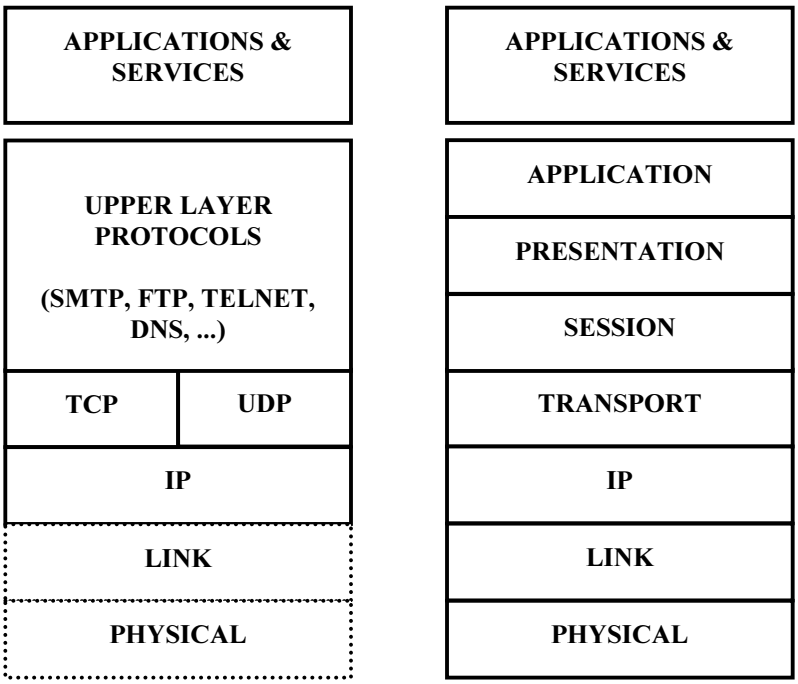


Abb. 1.2 Ein Überblick über die Internet-Protokolle

1.2.1 IP

Das hardwareunabhängige Internet-Protokoll (IP) [RFC791] kann auf verschiedene LAN- bzw. WAN-Protokollen aufgesetzt werden. IP kann der dritten OSI-Schicht zugeordnet werden. Es bietet einen verbindungslosen Zustelldienst unabhängig von dem darunterliegenden Medium an. Daten werden als Datagramme, eine Gruppe von Oktetten, gesendet. Dieser Dienst wird aber als unzuverlässig bezeichnet, da sowohl kein Kontrollmechanismus für die Korrektheit der empfangenen Datagramme existiert, als auch keine Korrektur fehlerhafter Daten erfolgt. Deswegen muß die Zuverlässigkeit von höheren Schichten gewährleistet werden.

Das IP übernimmt auch die Adressierung im Internet. Eine IP-Adresse ist 32 Bit groß und besteht aus einem Network-Identifier und einem Host-Identifier. Der Network-Identifier bezieht sich auf ein bestimmtes physikalisches Netzwerk im Internet, wobei der Host-Identifier ein bestimmtes Gerät bezeichnet, das an diesem Netzwerk angeschlossen ist.

Eine weitere Aufgabe dieses Protokolls ist das Routing. Dabei soll entschieden werden, welcher Weg für die Übertragung von Daten zwischen zwei Netzwerkgeräten gewählt werden soll.

1.2.2 TCP

Das Transmission Control Protocol (TCP) [RFC793] ist das verbindungsorientierte Transportprotokoll in der Internet-Protokollfamilie. TCP repräsentiert die Transportschicht des OSI-Modells und befindet sich über dem verbindungslosen IP.

Da TCP verbindungsorientiert ist, besteht eine Kommunikation zwischen zwei Geräten aus drei Phasen:

1. Aufbau der Verbindung

2. Datenübertragung

3. Abbau der Verbindung

Bei der Datenübertragung wird der Datenstrom vor dem Abschicken in sogenannte Segmente zerlegt und nach dem Empfang am Ziel wieder zusammengestellt. Alle Segmente werden dabei numeriert. Wenn ein Segment geschickt wird, startet die Quelle einen Zähler. Falls vor dem Ablaufen des Zählers keine Bestätigung über die erfolgte Segment-Übertragung aus dem Ziel kommt, wird dieser Vorgang wiederholt. Dadurch wird eine bestimmte Zuverlässigkeit gewährleistet. Eine mehrmalige Wiederholung kann aber auch zu einem schlechten Datendurchsatz führen.

1.2.3 UDP

Das User Datagram Protocol (UDP) ist das verbindungslose Transportprotokoll der Internet-Protokollfamilie. UDP gehört wie TCP zur Transportschicht und basiert genauso auf IP.

Da in UDP kein Verbindungsauf- und abbau erfolgt, ist UDP viel einfacher gestaltet als TCP. Ein UDP-Paket besteht nur aus einer Sender- und Empfänger-Portnummer, Prüfsumme, Längenangabe und natürlich aus Benutzerdaten. Dieser verbindungslose Aufbau führt aber zur Unzuverlässigkeit, da in UDP nicht sichergestellt wird, ob eine Datenübertragung erfolgreich abgeschlossen ist oder nicht. Die Korrektheit der Daten wird nur durch eine einfache Prüfsummenkontrolle überprüft. Die weiteren Prüfungen werden der Anwendungsschicht überlassen.

2. SNMP UND NETZMANAGEMENT

2.1 Einführung in das Netzmanagement

In den letzten Jahren gewinnen Rechnernetze insbesondere wegen dem Internet und dem Daten-Highway immer mehr an Bedeutung. Rechnervernetzung bringt viele Vorteile, wie die dezentrale Verarbeitung der Daten und verteilte Intelligenz, wodurch man viel an Sicherheit und Flexibilität gewinnen kann. Das Ressource-Sharing, mit dem teure und leistungsfähige Peripheriegeräte im Netzwerk vielen Benutzern kostengünstig bereitgestellt werden können, ist ein anderer Grund für den hohen Anstieg der Anzahl der Netze in den letzten Jahren.

Die Kommunikation zwischen den Rechnern birgt jedoch auch das Problem, die vielförmigen Komponenten zu verwalten und überwachen. Zur Aufgabe eines Netzmanagementprotokolls gehören die Konfiguration, Überwachung, Kostenzuordnung und Kontrolle der Netzwerkaktivitäten und Ressourcen. Die wachsende Komplexität und Vielfältigkeit der Netze erfordern standardisierte und herstellerunabhängige Werkzeuge für das Netzmanagement.

2.2 Komponente des Netzmanagements

Ein Netzmanagementsystem besteht aus vier Teilen [ROS95] (siehe Abb. 2.1):

- Mehreren verwalteten Knoten (Nodes)
- Mindestens einer Netzmanagement-Station (NMS), auf der das Netzmanagementprotokoll und die Applikationen für das Netzmanagement ablaufen
- Einem Netzmanagementprotokoll, das von der NMS und den verwalteten Knoten zum Austausch von Verwaltungsinformationen benutzt wird
- Verwaltungsinformationen

2.2.1 Verwaltete Knoten

Ein verwalteter Knoten kann ein physikalisches System, wie zum Beispiel ein Rechner (Host), Gateway, Mediagerät usw., oder ein logisches sein, wie zum Beispiel ein Dienst oder eine Netzwerk-Applikation.

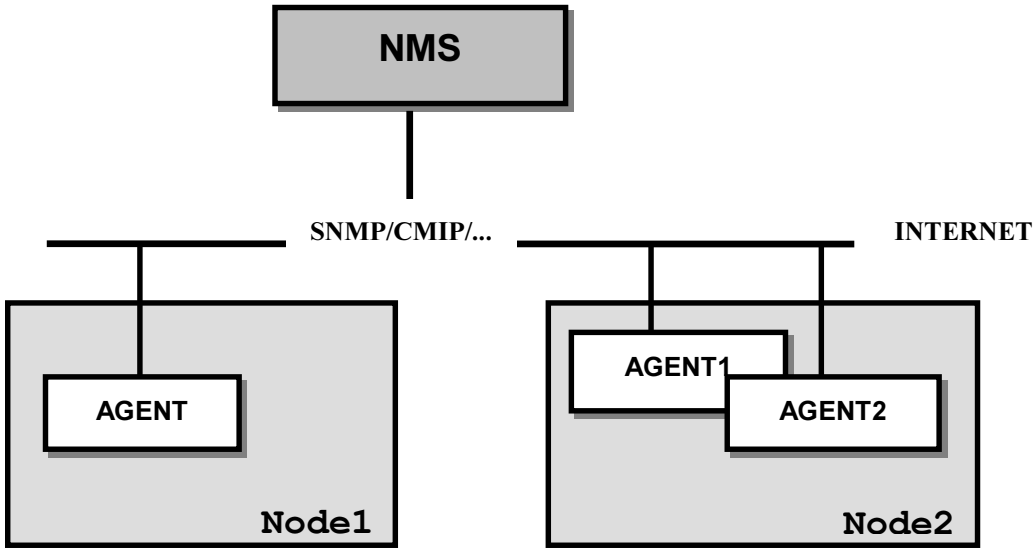


Abb. 2.1 Netzmanagmentsystem

Hauptsächlich bestehen die verwalteten Knoten, wie in der Abbildung 2.2 dargestellt, aus drei Komponenten [ROS95]:

1. Verschiedene Protokolle, die die anwendungsspezifischen Funktionen ausführen.

2. Ein Management-Protokoll, das die Beobachtung und Kontrolle des verwalteten Knotens gewährleistet.

3. Management Instrumentation, die die Kommunikation und Wechselwirkung zwischen den nützlichen Protokollen und dem Management-Protokoll ermöglicht. Diese Funktion wird normalerweise durch einen eigenen Kommunikationsmechanismus erreicht, bei dem die Daten der nützlichen Protokolle durch das Management-Protokoll zugegriffen und verändert werden können.

2.2.2 Netzmanagement-Station

Eine Netzmanagement-Station ist ein Rechner, auf dem folgendes abläuft [ROS93]:

- das Netzmanagement-Protokoll.

- die Netzmanagement-Anwendungen.

Das Netzmanagement-Protokoll ist für die Kommunikation zwischen einem oder mehreren Management-Stationen und den verwalteten Knoten verantwortlich. Die Netzmanagement-Anwendungen bestimmen die Aktionen bei der Verwaltung.

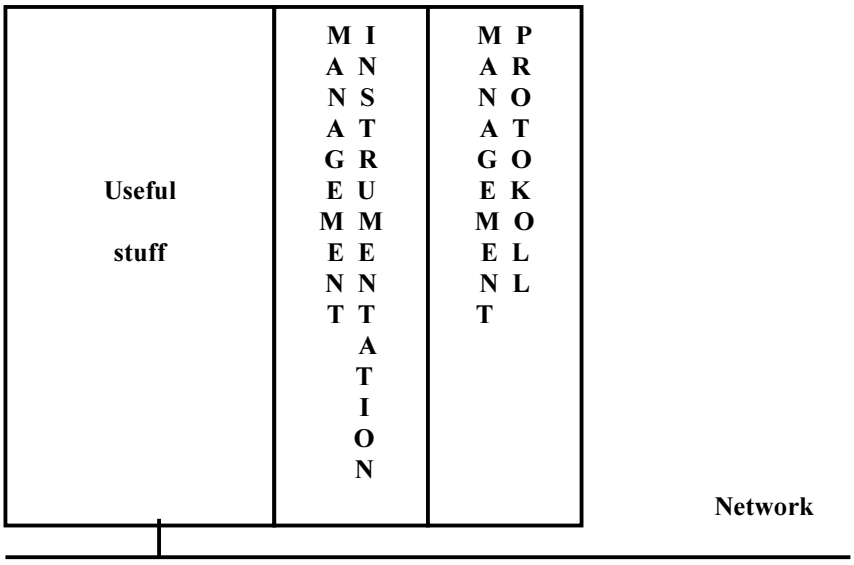


Abb. 2.2 Komponenten eines verwalteten Knotens [ROS95]

Bei einem Netzmanagementsystem muß das Hinzufügen der Netwerkverwaltung möglichst kleine Auswirkungen auf die verwalteten Knoten haben, so daß dieses System eine große Bandbreite von Knoten verwalten kann, ohne daß sie dadurch belastet wird. Deswegen übernimmt die Management-Station die Hauptverantwortung bei der Verwaltung. Die Verwaltungsarbeit verlagert sich auf diese Station. Dadurch kann man auch vermeiden, daß die Herstellungskosten von einfachen Knoten, wie Bridges, Routers usw., durch das Hinzufügen der Verwaltungsintelligenz enorm steigen.

Die Management-Stationen, die vom Bedienungspersonal eines großen Netzwerks betrieben werden, werden auch als Network Operations Center (NOC) bezeichnet.

2.2.3 Netzmanagement-Protokoll

In auf der Internet-Protokollfamilie basierenden Systemen wird es angenommen, daß jeder verwaltete Knoten mehrere Variablen besitzt. Durch das Lesen und Verändern dieser Variablen wird der Knoten beobachtet und kontrolliert. Das kann mit einem einfachen Protokoll verwirklicht werden.

Neben den Lese- und Schreiboperationen braucht man noch zwei andere Operationen [ROS93]:

- eine Durchlaufoperation, mit der die Management-Station feststellen kann, welche Variablen ein verwalteter Knoten unterstützt.

- Eine Operation, mit der ein verwalteter Knoten ein unerwartetes Ereignis der Management-Station informieren kann. Diese wird als "Trap" bezeichnet.

2.2.4 Netzmanagementinformation

Die Netzmanagementinformation besteht aus Daten, die in einer bestimmten kodierten Form über das Netzmanagement-Protokoll zwischen Management-Stationen und verwalteten Knoten ausgetauscht werden. Es wurde in dem letzten Kapitel erwähnt, daß man in dem Netzmanagement die verwalteten Knoten so betrachtet, als ob sie aus mehreren Variablen bestünden. Aus objektorientierter Sichtweise gesehen, verwendet man statt Variablen Objekte, die aus mehreren Variablen und Funktionen bestehen können. Wie die Struktur dieser Information aussehen soll, wird im Bezug auf SNMP in dem Kapitel 2.5 dargelegt.

2.3 SNMP

Mitte der 80er Jahre begann die Anzahl der mit dem Internet verbundenen Netze deutlich zu steigen. Damals gab es kein offenes (herstellerunabhängiges) Netzmanagementprotokoll, sondern jeder Hersteller entwickelte eigene Netzmanagementprodukte. Da aber ein Netzwerk meistens aus Geräten verschiedener Hersteller zusammengestellt wird, konnte man den hohen Aufwand für die Verwaltung solcher Netzwerke nicht mehr überwinden. Um dieses immer wachsende Problem beseitigen zu können, haben Entwickler im Jahre 1987 ein einfaches Protokoll, namens Simple Gateway Monitoring Protocol (SGMP), entwickelt. SGMP wurde in den nachfolgenden Jahren erweitert und 1989 unter dem Namen Simple Network Management Protocol (SNMP) standardisiert. Während SGMP nur für die Überwachung von Gateways geeignet ist, kann SNMP im Gegensatz zu seinem Vorgänger zum Netzmanagement beliebiger Netzmanagement Knoten benutzt werden.

Zur ungefähr gleichen Zeit hat eine andere Gruppe von Entwicklern versucht, das OSI-Modell für die Verwaltung der Netze einzusetzen. Das entworfene Protokoll wurde als Common Management Information Protocol (CMIP) bezeichnet. Außerdem wurde das Protokoll CMIP over TCP/IP (CMOT) speziell für Netzwerke, die die Internet-Protokollfamilie benutzen, als eine Abbildung des OSI-Netzmanagementprotokolls entwickelt.

In den nächsten Jahren hat die Organisation IAB (Internet Activities Board), die für die technische Entwicklung der Internet-Protokollfamilie verantwortlich ist, beschlossen, SNMP für kurzfristige Lösungen und CMOT für langfristige zu unterstützen.

Obwohl SNMP nach Meinungen der IAB in kürzester Zeit von CMOT ersetzt werden sollte, hat SNMP von seiner Beliebtheit bis jetzt nichts verloren und wurde ständig erweitert. SNMP wurde dank seiner Einfachheit weit verbreitet, während das viel komplexere Protokoll CMOT bis jetzt nur in wenigen Bereichen der Industrie implementiert wurde. Seit 1993 gibt es die Version 2 von SNMP, bei der die wesentlichen Schwachpunkte der ersten Version, wie die hohe Netzbelastung bei der Übertragung großer Mengen von Verwaltungsdaten, mangelhafte Sicherheit und fehlende Manager-Agenten-Hierarchie, beseitigt worden sind. Auch eine noch verbesserte dritte Version ist im Jänner 1998 standardisiert worden.

Alle Erklärungen beziehen sich zuerst nur auf die erste Version von SNMP (SNMPv1) [RFC1157]. Anschließend werden auch die Erweiterungen der zweiten (SNMPv2) [RFC1141, RFC1901, RFC1902, RFC1903, RFC1904, RFC1905, RFC1906, RFC1907, RFC1909,

RFC2011, RFC2012, RFC2013] und der dritten Version (SNMPv3) [RFC2271, RFC2573] behandelt.

2.4 SNMP im OSI-Modell

SNMP basiert auf UDP/IP. Der Grund, warum UDP an Stelle von TCP als Transportprotokoll verwendet wurde, ist die Einfachheit von UDP. UDP hat geringeren Anspruch auf Speicherplatz, CPU und andere Ressourcen als TCP. Einfache verwaltete Knoten, wie Repeater oder Modems, können UDP problemlos unterstützen. Dadurch verringert sich auch die Netzlast durch die Managementdaten [GIL95]. Als Nachteil kann die Unzuverlässigkeit bei der verbindungslose Datenübertragung auftreten. Die Korrektheit der Daten soll in diesem Fall durch höhere Schichten gewährleistet werden.

Wenn man die verschiedenen Elemente das Netzmanagement mit dem OSI-Schichtenmodell vergleicht, kann man SNMP der Kommunikationssteuerungsschicht zuordnen, während die darunterliegende UDP die Transport- und IP die Vermittlungsschicht darstellen (Abb. 3.3). Die LANs bzw. WANs bilden die untersten zwei Schichten, Sicherungs- und Bitübertragungsschicht. Die von SNMP benutzte Structure of Management Information (SMI) kann der Darstellungs- und die Management-Applikationen zusammen mit den einzelnen MIBs können der Anwendungsschichtschicht zugeordnet werden [MIL93].

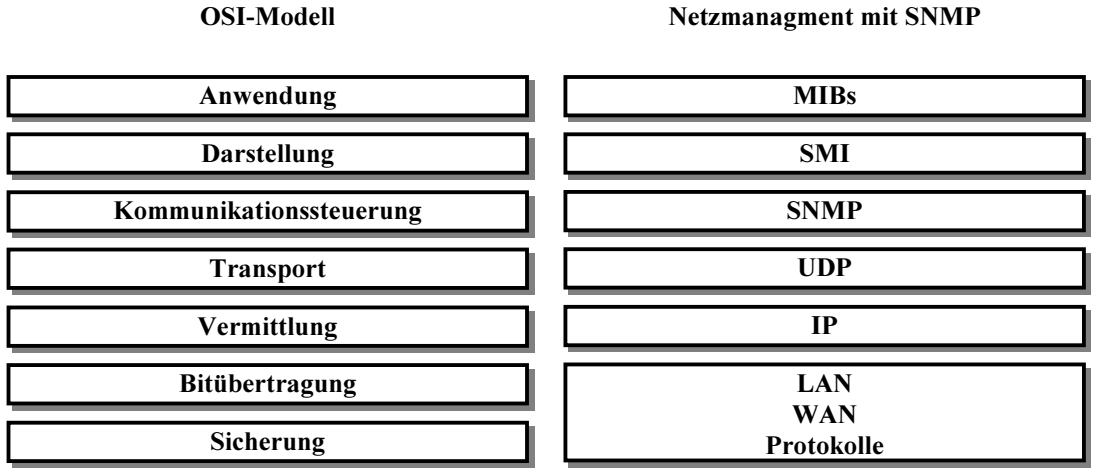


Abb. 2.3 Das OSI-Modell für SNMP

2.5 Structure of Management Information

Die verwalteten Objekte werden in einer Datenbank, namens Management Information Base (MIB), gespeichert. Die Structure of Management Information (SMI) bestimmt die Struktur der MIB. Die Beschreibung dieser Struktur erfolgt mit der OSI-Sprache Abstract Syntax Notation One (ASN.1) [GOR92]. Es wird aber nur ein Teil der Möglichkeiten von ASN.1

verwendet, um die Komplexität der Beschreibung möglichst gering zu halten. Die genaue Definition der SMI ist in [RFC1155] erklärt.

Jedes verwaltete Objekt ist mit einem ASN.1-Makro, das als `OBJECT-TYPE`-Makro benannt wird, dargestellt. Die Syntax dieses Makros sieht folgendermaßen aus:

```
OBJECT-TYPE MACRO ::=
  BEGIN
    TYPE NOTATION ::= "SYNTAX" type (TYPE ObjectSyntax)
                      "ACCESS" Access
                      "STATUS" Status
    VALUE NOTATION ::= value (VALUE ObjectName)

    Access ::= "read-only"
                    | "read-write"
                    | "write-only"
                    | "not-accessible"
    Status ::= "mandatory"
                    | "optional"
                    | "obsolete"
  END
```

Der Zugriff auf ein verwaltetes Objekt kann `read-only`, `read-write`, `write-only` oder `not-accessible` sein.

Der `Status` zeigt die Notwendigkeit eines verwalteten Objektes. Wenn ein Objekt den Status `mandatory` hat, muß es unbedingt implementiert werden. Objekte mit der Status `optional` können von verwalteten Knoten implementiert werden, müssen aber nicht. Der Status `obsolete` zeigt, daß man dieses Objekt nicht mehr zu implementieren braucht.

Die `ObjektSyntax` beschreibt den abstrakten Datentyp jedes verwalteten Objekts. Der tatsächliche Datentyp ist einer der Typen, die in einer CHOICE in ASN.1 definiert sind. Es gibt drei Arten von Wahlmöglichkeiten [ROS93]:

1. `simple`, bezieht sich auf die vier einfachen ASN.1-Typen:

   - `INTEGER`: Ein Datentyp, der eine natürliche Zahl als Wert hat. Man kann auch symbolische Namen für Werte vergeben, die Instanzen dieses Datentyps annehmen können, z.B.:

     ```
     Status ::=
          INTEGER { up(1), down(2), testing(3) }

     myStatus Status ::= up      - oder 1
     ```

     Es ist nur kein Bezeichner mit dem Wert Null erlaubt, da dies zum Erkennen von allgemeinen Verpackungsfehlern mit ganzzahligen Werten dient.

   - `OCTET STRING`: Ein Datentyp, der kein oder mehrere Oktette als Wert hat, die einen Wert zwischen 0 und 255 haben können.

- `OBJECT IDENTIFIER`: Ein Datentyp, der ein eindeutig festgelegtes Objekt identifiziert. Auf Grund seiner komplizierten Semantik und seiner Wichtigkeit für das Netzmanagement, wird er am Ende dieses Abschnitts ausführlicher erläutert.

- `NULL`: Ein Datentyp, der als Platzhalter dient.

2. `application-wide`, beruht auf sechs speziellen Datentypen [RFC1155]:

   - `IpAdress`: Mit diesem Datentyp wird eine 32-Bit-IP-Adresse gezeigt.

   - `NetworkAdress`: Mit diesem Datentyp wird eine Netzwerkadresse aus einer von möglichen Protokollfamilien dargestellt. Momentan ist nur die Internet-Protokollfamilie in `CHOICE` von diesem Datentyp definiert. Deswegen ist der Datenyp `NetworkAdress` mit dem `IpAdress` gleichgestellt.

   - `Counter`: Dieser Datentyp stellt eine nicht negative Zahl dar, die monoton wächst, bis sie die maximale Zahl 2^{32}-1 erreicht und dann wieder bei Null anfängt.

   - `Gauge`: Dieser Datentyp dient zur Darstellung einer nicht negativen Zahl, die inkrementiert und dekrementiert werden kann, die aber bei der maximalen Zahl 2^{32}-1 festgehalten wird.

   - `TimeTicks`: Mit diesem Datentyp kann eine nicht negative Zahl beschrieben werden, die seit einem Ereignis die Zeit in hundertstel Sekunden zeigt. Der Startpunkt muß bei der Definition festgelegt werden.

   - `Opaque`: Dieser Datentyp stellt eine beliebige Verpackung dar. Mit Hilfe dieses Datentyps können Daten in einer Zeichenfolge von Oktetten verpackt werden.

3. `simply constructed`, bezieht sich auf zwei zusammengesetzten ASN.1-Typen:

   - list: Dieser Datentyp ermöglicht das Errichten von geordneten Listen mit keinem oder mehreren Elementen, die aus anderen ASN.1-Typen stammen. Seine Syntax hat die Form:

     ```
     <list> ::= SEQUENCE { <type1>, ..., <typeN> }
     ```

   - table: Mit diesem Datentyp kann man eine geordnete Liste mit keinem oder mehreren Elementen desselben ASN.1-Typs bilden. Die Syntax von diesem Typ lautet:

     ```
     <table> ::= SEQUENCE OF <entry>
     ```

Die SMI-Definition in [RFC1155] sieht folgendermaßen aus:

```
RFC1155-SMI DEFINITIONS ::= BEGIN

EXPORTS -- EVERYTHING
```

```
  internet, directory, mgmt,
  experimental, private, enterprises,
  OBJECT-TYPE, ObjectName, ObjectSyntax, SimpleSyntax,
  ApplicationSyntax, NetworkAddress, IpAddress, Counter,
  TimeTicks, Opaque;

-- the path to the root

internet     OBJECT IDENTIFIER ::= { iso org(3) dod(6) 1 }

directory    OBJECT IDENTIFIER ::= { internet 1 }

mgmt         OBJECT IDENTIFIER ::= { internet 2 }

experimental OBJECT IDENTIFIER ::= { internet 3 }

private      OBJECT IDENTIFIER ::= { internet 4 }

enterprises  OBJECT IDENTIFIER ::= { private 1 }

-- definition of object types

OBJECT-TYPE MACRO ::=
BEGIN
TYPE NOTATION ::= "SYNTAX" type (TYPE ObjectSyntax)
                  "ACCESS" Access
                  "STATUS" Status
VALUE NOTATION ::= value (VALUE ObjectName)

Access ::= "read-only"
            | "read-write"
            | "write-only"
            | "not-accessible"
Status ::= "mandatory"
            | "optional"
            | "obsolete"
END

-- names of objects in the MIB

ObjectName ::=
  OBJECT IDENTIFIER

-- syntax of objects in the MIB

ObjectSyntax ::=
  CHOICE {
          SIMPLE
                  SimpleSyntax,

-- note that simple SEQUENCEs are not directly mentioned
-- here to keep things simple (i.e., prevent mis-use).
-- However, application-wide types which are IMPLICITly
-- encoded simple SEQUENCEs may appear in the following
-- CHOICE
```

```
            application-wide
                 ApplicationSyntax
 }

SimpleSyntax ::=
  CHOICE {
            number
                 INTEGER,
            string
                 OCTET STRING,
            object
                 OBJECT IDENTIFIER,
            empty
                 NULL
}

ApplicationSyntax ::=
  CHOICE {
            address
                 NetworkAddress,
            counter
                 Counter,
            gauge
                 Gauge,
            ticks
                 TimeTicks,
            arbitrary
                 Opaque

-- other application-wide types, as they are
-- defined, will be added here
}


-- application-wide types

NetworkAddress ::=
  CHOICE {
            internet
                 IpAddress
}

IpAddress ::=
  [APPLICATION 0]          -- in network-byte order
    IMPLICIT OCTET STRING (SIZE (4))

Counter ::=
  [APPLICATION 1]
    IMPLICIT INTEGER (0..4294967295)

Gauge ::=
  [APPLICATION 2]
    IMPLICIT INTEGER (0..4294967295)

TimeTicks ::=
  [APPLICATION 3]
```

```
      IMPLICIT INTEGER (0..4294967295)

  Opaque ::=
    [APPLICATION 4]          -- arbitrary ASN.1 value,
      IMPLICIT OCTET STRING   --   "double-wrapped"

  END
```

Object Identifier

Jedes verwaltete Objekt wird durch einen `OBJECT IDENTIFIER` (OID) eindeutig identifiziert. Ein `OID` ist eine Folge von nicht negativen ganzzahligen Werten, die einen Baum durchlaufen, der eine Wurzel hat, die mit einer Menge von gekennzeichneten Knoten (labeled nodes) verbunden ist. Jeder Knoten kann weitere untergeordnete Knoten besitzen, die man als Unterbäume dieses Knotens bezeichnet.

Die Wurzel selbst hat kein Kennzeichen (label), besitzt aber drei Unterbäume:

1. `ccitt (0)`: dieser wird von Comité Consultatif International Télégraphique et Téléphonique (CCITT) verwaltet.

2. `iso (1)`: dieser wird von International Organization for Standardization und International Electrotechnical Commitee (ISO/IEC) verwaltet.

3. `joint-iso-ccitt (2)`: dieser wird von ISO/IEC und CCITT gemeinsam verwaltet.

Für das Netzmanagement ist nur der zweite Unterbaum `iso` von Bedeutung. Dieser hat vier weitere Unterbäume (siehe Abb. 2.4):

1. `standard (0)`: Dieser Unterbaum enthält einen Unterbaum für jeden internationalen Standard.

2. `registration-authority (1)`: Dieser Unterbaum ist für die OSI-Meldebehörden reserviert.

3. `member-body (2)`: Dieser hat einen Unterbaum für jede Mitgliedsgruppe von ISO/IEC. Der Wert des zugeordneten Bezeichners für jeden Knoten ist gleich dem dezimalen Ländercode (DCC).

4. `org (3)`: Jede von ISO/IEC unterstützte Organisation bekommt einen eigenen Unterbaum unter diesem zugewiesen. Die U.S. Department of Defense hat auch einen Unterbaum, nämlich `dod (6)`. Unter diesem Baum gibt es der Unterbaum für Internet mit der Identifikationsnummer 1.

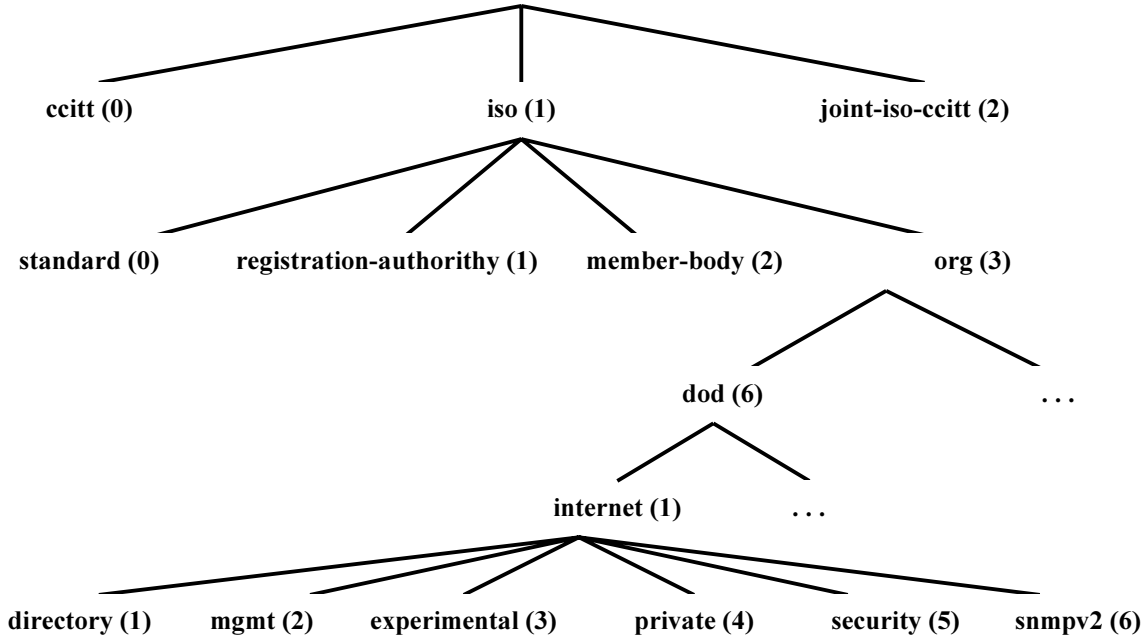


Abb. 2.4 Der Baum für OBJECT IDENTIFIERS von Internet

Der `internet` Unterbaum hat derzeit 6 Unterbäume. Drei von denen, nämlich `mgmt`, `experimental` und `private`, werden jetzt näher betrachtet. Der Unterbaum `mgmt` (management) wird von Internet Assigned Number Authority verwaltet und enthält alle standardisierten MIBs. Neue und noch nicht erprobte Verwaltungsobjekte werden unter dem Unterbaum `experimental` aufbewahrt. Falls sie als geeignet für das Netzmanagement erwiesen werden, können sie mit einer Zustimmung von Internet Assigned Number Authority als ein internationaler Standard von dem Unterbaum `experimental` zu dem `mgmt` verschoben werden. Der Unterbaum `private` ermöglicht den Herstellern ihre eigene Objekte zu definieren. Eine Verschiebung in den `mgmt`-Unterzweig würde diese Objekte weltweit gültig machen, falls sie natürlich international standardisiert werden.

Den OID eines Objekts kann man als eine Liste aller Zahlen beschreiben, die beim Durchlaufen des Baumes aller Komponenten von der Wurzel in Richtung zu dem gewünschten Objekt gefunden werden. Die Zahlen werden in der punktierten Schreibweise ausgegeben. Jeder Zahl ist auch ein Name zugeordnet. An Stelle der Zahlenwerte kann man auch diese Textwerte verwenden. Die Syntax des OIDs sieht dann folgendermaßen aus [PER93]:

```
"{" {{<Name>["("<Zahl>")"]} | <Zahl>}... "}"
oder
<Zahl> ["."<Zahl>]...

wobei  <Name> ein Objektname und
       <Zahl> die Kennzahl eines Objekts ist
```

Für den OID des Unterbaums `private (4)` kann zum Beispiel eine der folgenden Beschreibungsformen benutzt :

```
{ iso org(3) dod(6) internet(1) private(4) }

oder 1.3.6.1.4

oder {internet 4}
```

Um eine Variable innerhalb eines Objekts identifizieren zu können, braucht man dem OID des Objekts die Kennzahl der Variable innerhalb des Objekts anhängen. Bei einfachen Variablen bedeutet das, daß man dem Ende des OIDs des jeweiligen Objekts ".0" hinzufügen muß. Bei Tabelleneinträgen könnte hinter dem OID des Objekts eine Zahl als Spaltennummer, einen String bestimmter Länge oder unbestimmter Länge, eine IP-Adresse, eine Netzwerk-Adresse oder einen anderen OID stehen.

2.6 Management Information Base

Alle Objekte, die von verwalteten Knoten implementiert werden sollten, werden in einer hierarchisch aufgebauten Datenstruktur, als Management Information Base (MIB) bezeichnet, gespeichert.

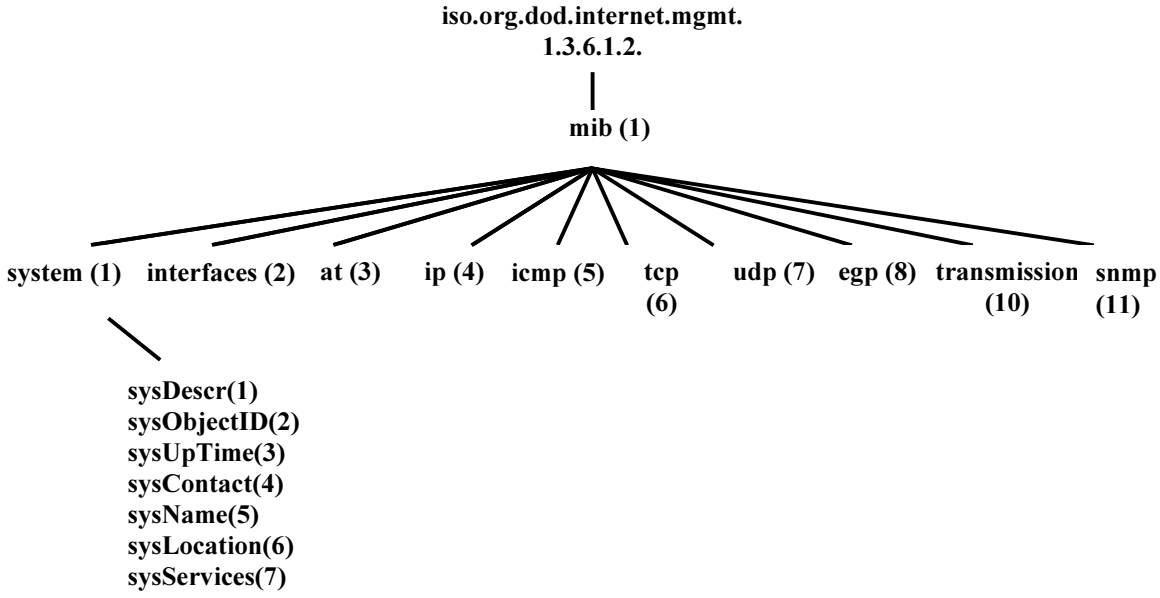


Abb. 2.5 Die MIB-II Struktur

Die erste MIB für die Internet-Protokollfamilie, genannt MIB-I, ist in [RFC1156] beschrieben. Im OID-Baum wurde sie unter den `mgmt`-Unterbaum plaziert und bekam den OID 1.3.6.1.2.1. Der Hauptgedanke bei der Bildung dieser MIB-I war das, daß sie möglichst wenige verwaltete Objekte enthalten sollte. Sie enthält insgesamt 114 Objekte, die in acht Gruppen unterteilt sind.

Wenn ein verwalteter Knoten eine Funktion aus einer von diesen Gruppen braucht, dann muß er alle Objekte dieser Gruppe unterstützen.

Die MIB-I wurde im Jahre 1991 von der MIB-II ersetzt. Die neue MIB-II wurde als eine verbesserte Obermenge seiner Vorgänger in [RFC1213] veröffentlicht. MIB-II beinhaltet außer den alten MIB-I-Objekten noch 57 zusätzliche Verwaltungsobjekte. Die dadurch insgesamt 171 gewordenen Objekte wurden in 10 Gruppen gegliedert. Die Abbildung 2.5 zeigt die MIB-II-Struktur im OID-Baum. Als Beispiel wurden auch die Komponenten des `system`-Unterbaums dargestellt. Die Objekte, die nur in MIB-II definiert sind, werden *kursiv* geschrieben.

Im folgenden werden die Gruppen der MIB-II teilweise mit Hilfe von Case-Diagrammen erläutert. Für die visuelle Darstellung des Flusses von Verwaltungsinformation in einer Schicht wurde das Case-Diagramm als nützliches Hilfsmittel entwickelt (siehe Abb. 2.6). In einer Schicht ist die Zahl der aus der darunterliegenden Schicht empfangenen Pakete gleich der Zahl der fehlerhaften Pakete (`InErrors`) plus die Zahl der innerhalb dieser Schicht weitergeleiteten Pakete (`forwPackets`) und die Zahl der zu der darüberliegenden Schicht weiter geschickten Pakete (`inDelivers`). Die Zahl der zur darunterliegenden Schicht geschickten Pakete ist gleich der Zahl der aus der darüberliegenden Schicht empfangenen Daten (`outRequests`) zusammen mit der Zahl der innerhalb dieser Schicht weitergeleiteten Pakete (`forwPackets`). In Case-Diagrammen werden auch dargestellt, wo Zähler erhöht werden (siehe Abb. 2.6).

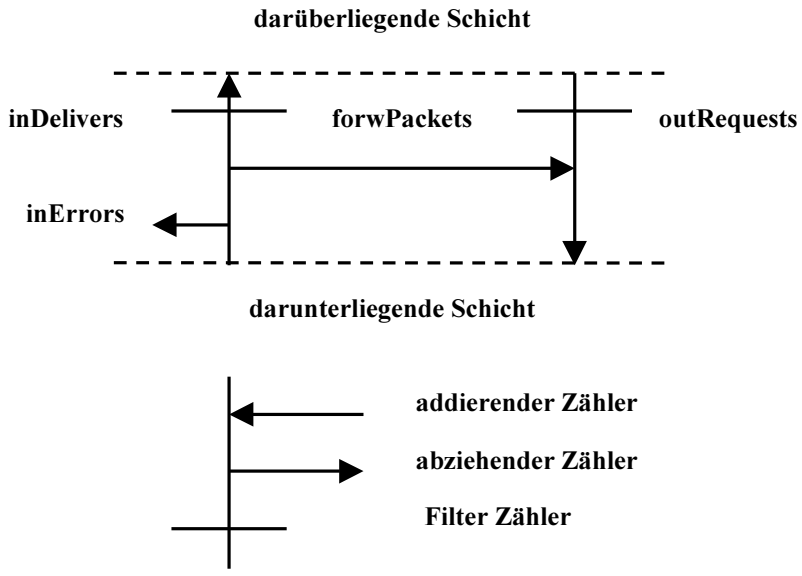


Abb. 2.6 Case-Diagramm [ROS93]

Die MIB-II hat folgende Gruppen:

- `system`: Diese Gruppe beinhaltet Informationen über die Konfiguration des verwalteten Knotens, wie eine Systembeschreibung (`sysDescr`),

eine Bezeichnung des Herstellers (`sysObjectID`), die Zeit, wie lange die Software in Betrieb ist (`sysUpTime`), den Namen der Kontaktperson (`sysContact`), den Namen des Geräts (`sysName`), den Aufstellungsort des Geräts (`sysLocation`) und die durch das Gerät angebotenen Dienste (`sysService`). Sie muß von allen verwalteten Knoten implementiert werden.

- `interfaces:` Diese Gruppe muß von allen verwalteten Knoten implementiert werden. Sie enthält 23 Objekte zur Verwaltung der Informationen über die Hardware-Schnittstellen in einem verwalteten Knoten. In der Abbildung 3.7 wird ein Case-Diagramm für diese Gruppe abgebildet.

- `at`: Diese Adress-Übersetzungsgruppe wurde in der MIB-II als "deprecated" markiert. Das heißt, daß die Gruppe nicht mehr zu implementieren ist und sich in der MIB-II nur aus Kompatibilitätsgründen mit der MIB-I befindet. Sie beinhaltet eine Tabelle, die zur Zuordnung von IP-Adressen in physikalische Adressen dient.

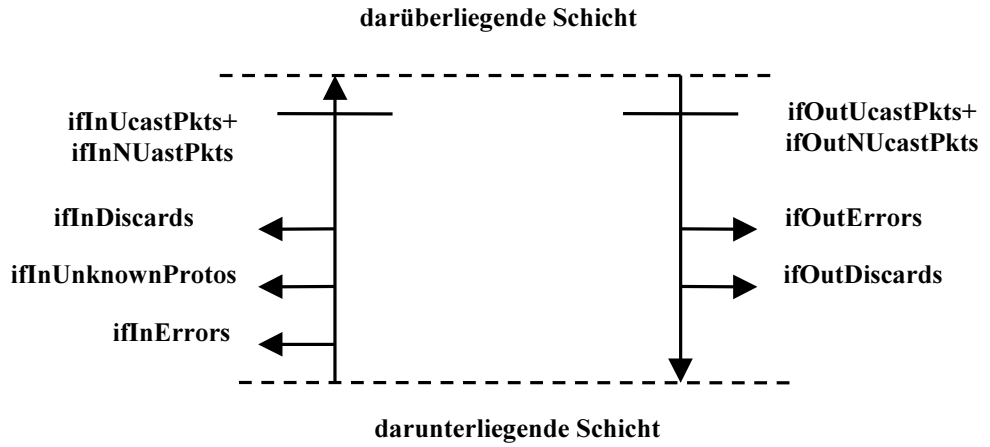


Abb. 2.7 Case-Diagramm für die `interface`-Gruppe [ROS93]

- `ip`: Die IP-Gruppe, die die Informationen über das Internet Protokoll (IP) beinhaltet, ist zwingend notwendig für alle verwalteten Knoten. In dieser Gruppe gibt es Objekte, die zur Verwaltung statistischer Werte bezogen auf IP dienen, und drei Tabellen, eine für die Verwaltung der IP-Adresse des Knotens (`ipAddrTable`), eine zweite für die Zuordnung von physikalischen zu IP-Adressen (`ipNetToMediaTable`) und eine dritte für die Routingfunktionen (`ipRoutingTable`). Die Abbildung 2.8 stellt ein Case-Diagramm für diese Gruppe dar.

- `icmp`: Diese Gruppe, die von allen verwalteten Knoten implementiert werden muß, beinhaltet 26 Zähler. Für jeden ICMP-Nachrichtentyp gibt es zwei Zähler [ROS93]. Einer zählt, wie oft dieser Nachrichtentyp von der eigenen IP-Einheit erzeugt wurde, während der andere die Anzahl der von der eigenen IP-Einheit empfangenen Nachrichtentypen zählt. Es gibt vier weitere Zähler, die jeweils die Anzahl der ICMP-Nachrichten, die empfangen, gesendet, fehlerhaft empfangen oder wegen eines Fehlers nicht gesendet wurden, zählen.

- `tcp`: Diese Gruppe muß von allen verwalteten Knoten, die TCP unterstützen, implementiert werden. Die Gruppe verwaltet statistische Informationen für TCP und beinhaltet eine Tabelle, die Auskunft über die TCP benutzenden Anwendungseinheiten gibt (`tpcConnTable`). In der Abbildung 2.9 wird ein Case-Diagramm für diese Gruppe gezeigt.

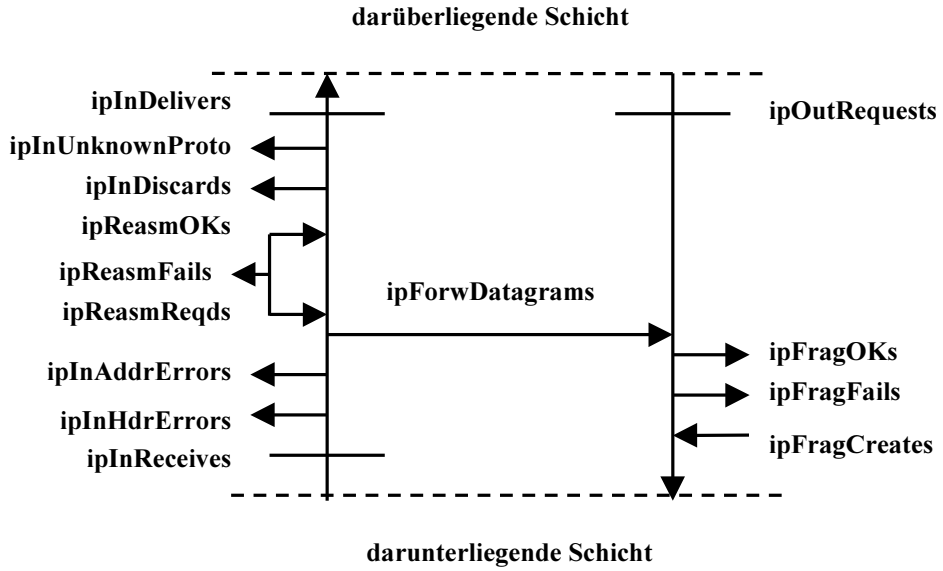


Abb. 2.8 Case-Diagramm für die `ip`-Gruppe [ROS93]

- `udp`: Die udp-Gruppe ist für alle verwalteten Knoten vorgeschrieben, die UDP unterstützen. Sie beinhaltet vier Zähler und eine Tabelle, die für die Zuordnung von dem lokalen UDP-Port zu der eigenen IP-Adresse benutzt wird.

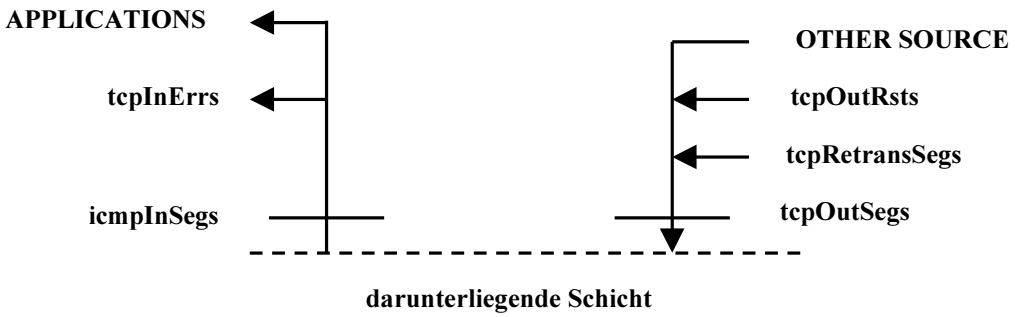


Abb. 2.9 Case-Diagramm für `tcp`-Gruppe [ROS93]

- `egp`: Diese Gruppe muß von allen verwalteten Knoten unterstützt werden, die das Exterior Gateway Protocol implementiert haben.

- *`transmission`*: Diese Gruppe dient in der MIB-II als Platzhalter für medienabhängige MIBs.

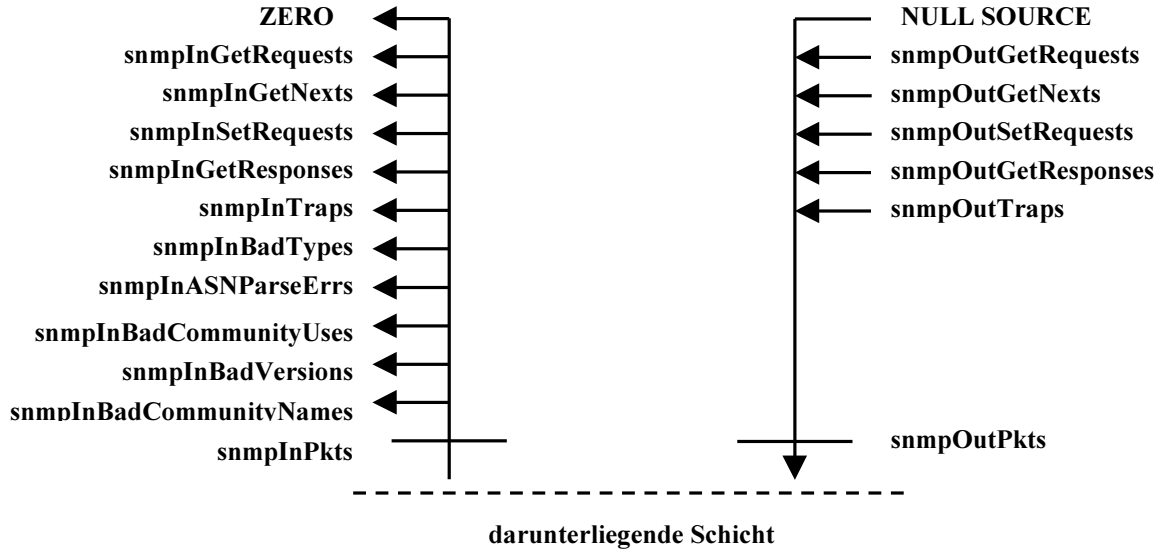


Abb. 2.10 Case-Diagramm für die `snmp`-Gruppe [ROS93]

- *`snmp`*: Die *`snmp`*-Gruppe, die neu in der MIB-II definiert wurde, verwaltet statistische Informationen über SNMP-Pakete. Die Abbildung 2.10 zeigt das Case-Diagramm für diese Gruppe. Die Anzahl der eingelangten SNMP-Nachrichten und der auftretenden Fehler beim Senden und Empfangen von Paketen, wie zum Beispiel zu lange PDUs (`snmpInTooBigs`, `snmpOutTooBigs`), ungültige Namen (`snmpInNoSuchName`, `snmpOutNoSuchName`) usw., werden gezählt.

2.7 Operationen von SNMPv1

In SNMP unterscheidet man zwischen Management-Stationen (Managers) und Agenten. Auf der Management-Station laufen das Netzmanagement-Protokoll und für das Netzmanagement relevante Applikationen ab. Der Agent bildet die Schnittstelle zwischen der Management-Station und dem verwalteten Knoten. In SNMPv1 gibt es vier Operationen, die zum Austausch von Verwaltungsinformationen zwischen Management-Stationen und Agenten dienen:

1. `GetRequest`: Mit dieser Anforderung kann ein Manager-Prozeß eine bestimmte Verwaltungsinformation aus der MIB des verwalteten Knotens erhalten.

2. `GetNextRequest`: Diese Nachricht wird vom Manager-Prozeß benutzt, um das lexikographische nächste Element im Namensbaum auszulesen.

3. `SetRequest`: Hiermit kann der Manager-Prozeß die Verwaltungsinformation in einem Agenten verändern.

4. `GetResponse`: Mit diesem Befehl liefert der Agent das Ergebnis einer Get-, GetNext- oder SetRequest-Aktion oder eine Fehlermeldung, falls die gewünschte Anforderung nicht durchgeführt werden konnte, zurück.

5. `Trap`: Mit dieser Nachricht kann ein Agent eine Management-Station über unerwartete Ereignisse benachrichtigen.

Um eine einfache Variable zu lesen, muß man mit dem `GetRequest`-Befehl auf die Adresse dieser Variable zugreifen, die durch das Anhängen einer ".0" ans Ende des jeweiligen Objektnames gebildet wird. Mit einem darauffolgenden `GetNextRequest` wird auf das lexikographisch nächste Element im MIB-Baum zugegriffen. Als Beispiel könnte ein Zugriff auf einfache Variablen aus der `system`-Unterbaum folgendermaßen aussehen:

```
system-MIB     ..............   1.3.6.1.2.1.1

get(system-MIB.sysUpTime.0)       liefert z.B. 55908844
                                  (6 Tage, 11 h, 18 min, 8.44 s)

getnext(system-MIB.sysUpTime.0)   liefert MELIH AYAL

get(system-MIB.sysContact.0)      liefert auch MELIH AYAL

getnext(system-MIB.sysContact.0)  liefert PC91

getnext(system-MIB.sysContact)    liefert MELIH AYAL
```

Mit einem einzigen `GetRequest`- oder `GetNextRequest`-Befehl kann ein Management-Prozeß mehrere Werte aus dem MIB eines Agenten herauslesen. Falls aber ein Fehler beim

Lesen eines Wertes auftritt, werden die anderen, richtig gelesenen, Werte in der `GetResponse`-Meldung undefiniert. Wenn der Manager nach vielen Werten in einer `GetRequest`-Operation fragt, könnten alle Informationen aus dieser Aktion nur wegen eines Fehlers Zunichte gemacht werden. Aber wenn er in jeder Operation nur einen Wert ausliest, muß er mehrere unabhängige Anforderungen schicken, was aber zu einem stark erhöhten Verkehr auf dem Netz führt.

Anstatt die Werte mit dem `GetRequest`-Befehl zu lesen, kann man mit dem `GetNextRequest`-Operator arbeiten, der von einer Objektinstanz zur nächsten weitergeht. Damit kann man eine Tabelle oder die gesamte MIB durchsuchen. Eine Fehlermeldung wird dann ausgegeben, wenn das nächste Element lexikographisch größer oder gleich dem lexikographisch größten Instanzenbezeichner im Community-Profile, eine Liste der für diese Verwaltungsrahmen zugelassenen Agenten und Manager, ist [ROS93].

Bei Tabelleneinträgen wird dem Tabellennamen ein das gesuchte Element eindeutig identifizierender Ausdruck anhängt. Da bei dem `GetNextRequest`-Befehl die Adresse des gelesenen Elements auch mitgeliefert wird, kann der Manager sofort das Ende der Tabelle erkennen, wenn der Prefix der zurückgelieferten Adresse nicht mehr mit dem gewünschten Tabellennamen übereinstimmt. Ein Zugriff auf Tabelleneinträgen könnte folgendermaßen erfolgen:

```
Interfaces-MIB   ......... 1.3.6.1.2.1.2

get(Interfaces-MIB.2.ifIndex.1)        liefert die Nummer des
                                       ersten Interfaces

getnext(Interfaces-MIB.2.ifIndex.1)    liefert die Nummer des
                                       zweiten Interfaces

get(Interfaces-MIB.2.ifIndex.2)        liefert auch die Nummer
                                       des zweiten Interfaces
```

Der `SetRequest`-Befehl dient zum Verändern der Einträge, aber auch zum Hinzufügen und Löschen der Werte. Da sehen wir wieder die vorteilhafte Einfachheit von SNMP gegenüber OSI-CMIP. Während in CMIP für diese Operationen drei verschiedene Befehle, nämlich `action`, `create` und `delete`, benötigt werden, braucht man hier nur den `SetRequest`-Befehl.

Durch Trap-Meldungen benachrichtigt ein Agent einem oder mehreren Manager über unerwartete Erreignisse, wie die Neuinitialisierung (`coldStart(0)`) bzw. Zurücksetzung des Agenten (`warmStart(1)`), unterbrochene Kommunikationsverbindung (`linkdown(2)`), neue Verbindung (`linkUp(3)`), ungültiger Community-Name (`authentificationFailure(4)`), unerreichbare Nachbarstation (`egpNeighborLoss(5)`) und herstellerspezifische Fehler (`enterpriseSpecific (6)`) [GIL95].

2.8 SNMP Protocol Data Unit (PDU)

In der Abbildung 3.11 wird ein allgemeiner Übertragungsrahmen von SNMP gezeigt. Der erste und letzte Teil beinhaltet die für die LAN- bzw. WAN-Protokollen benötigten Informationen. Diesen folgen IP- und UDP-Kopf.

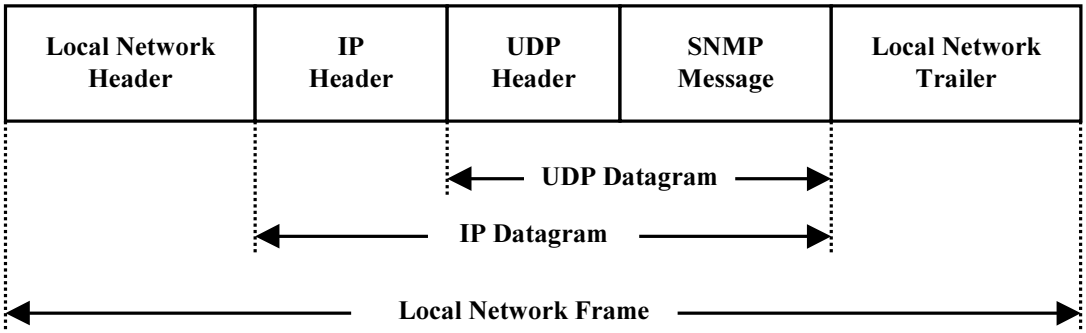


Abb. 2.11 SNMP Nachricht in einer Übertragungsrahmen [MIL93]

Dem UDP-Kopf folgt die SNMP-Nachricht, die hauptsächlich aus drei Teilen besteht (Abb. 3.12). Der erste Teil beinhaltet die Versionsnummer. Falls eine Nachricht mit einer unbekannten Versionsnummer empfangen wird, wird sie einfach weggeworfen.

Die Zusammengehörigkeit der Agenten und Manager wird über eine sogenannte Gemeinschaft (Community) geregelt. Da kann ein Manager nur mit einem Agenten kommunizieren, falls die beiden der selben Gemeinschaft angehören. Als Beweis dafür wird in einer Nachricht auch der Gemeinschaftsname übertragen. Falls eine Nachricht mit einem unbekannten Gemeinschaftsnamen empfangen wird, wird sie als ungültig betrachtet.

Abb. 2.12 Struktur einer SNMP-Nachricht

Der PDU-Teil einer SNMP-Nachricht für `Get-`, `GetNext-`, `SetRequest-` und `GetResponse`-Befehle besteht aus folgenden Teilen (Abb. 2.13):

- PDU-Typ: Der PDU-Typ wird als eine Zahl, nämlich eine "0" für den `GetRequest-`, "1" für den `GetNextRequest-`, "2" für den `GetResponse-`, "3" für den `SetRequest-` und "4" für den `Trap-`Befehl, angegeben.

- Anfragenummer: Die Anfragenummer (request-id) wird vom Manager dazu benutzt, zwischen den herankommenden Anfragen zu unterscheiden.

- Fehler-Status: Falls ein Fehler während der Durchführung der erwünschten Aktion auftritt, wird hier die Art des Fehlers eingetragen. Ansonsten bleibt der Status gleich Null.

- Fehler-Index: Falls Fehler-Status nicht Null, zeigt er an, bei welcher Instanz der Fehler aufgetreten ist.

- Nutzdaten: Nutzdaten bestehen aus einer Liste von Variablen mit je einem Namen und Wert (variable bindings). Die Werte der Variablen sind für die GetRequest- und GetNextRequest-Operationen irrelevant.

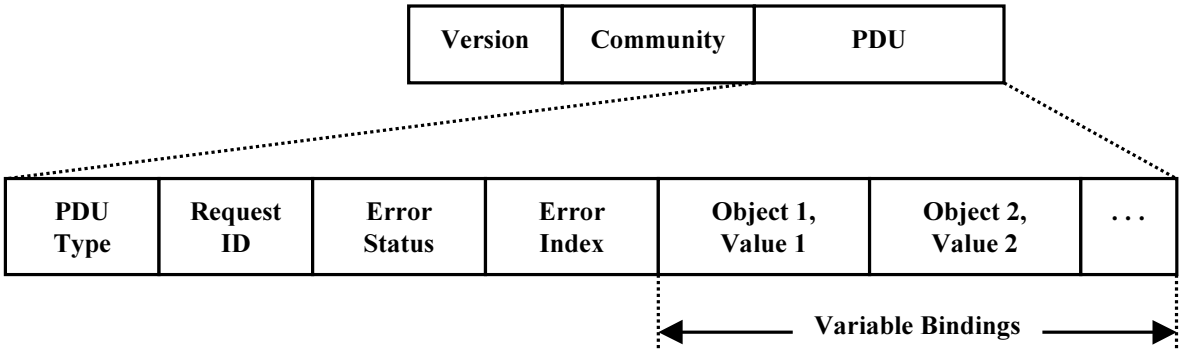


Abb. 2.13 PDU-Struktur für Get-, GetNext-, SetRequest und GetResponse-Befehle

In der Trap-PDU gibt es außerdem folgende Felder (Abb. 2.14):

- enterprise: Der Wert von sysObjectID des Agenten wird hier eingetragen.

- agent-addr: Die Netzwerkadresse des Agenten wird in dieses Feld geschrieben.

- generic-trap: Hier wird eingetragen, was für ein unerwartetes Ereignis aufgetreten ist.

- specific-trap: Falls es sich um einen herstellerabhängigen Trap handelt, gibt dieses Feld die herstellerspezifische Bezeichnung dafür aus.

- time-stamp: Hier wird der Wert des MIB-Objekts sysUpTime des Agenten, als dieses Ereignis passierte, geschrieben.

- variable-bindings: Dieses Feld besteht aus einer Liste von Variablen, die die Informationen über das Ereignis enthält.

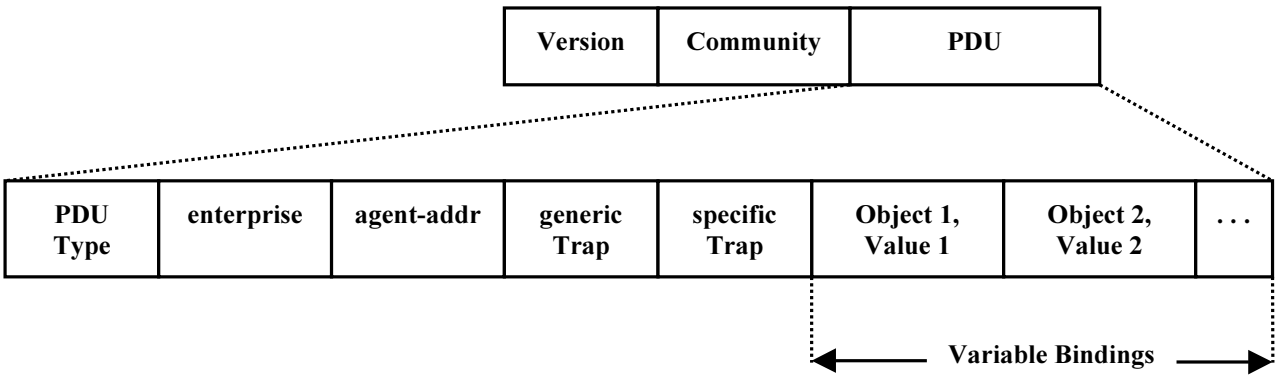


Abb. 2.14 PDU-Struktur für Trap-Meldungen

2.9 SNMPv2

Beim praktischen Einsatz von SNMPv1 wurden verschiedene Schwachpunkte dieses Protokolls offensichtlich. Aus diesem Grund wurden einige Verbesserungen im Bereich der Informationssicherheit sowie neue Erweiterungen für die Kommunikation zwischen verschiedenen Managern vorgenommen und das resultierende Protokoll wurde 1993 als SNMP Version 2 (SNMPv2) standardisiert.

Die erste Version von SNMP wurde oft wegen mangelndes Sicherheitskonzept kritisiert. Zur Erhöhung der Sicherheit wurde die Community-Struktur durch die neue Party-Struktur ersetzt, über welche die Verteilung von Zugriffsrechten sowie die Verschlüsselung der zu übertragenden Nachrichten möglich ist. Jeder verwaltete Knoten enthält eine Party, die aus einer Identifikationsnummer, einem Transportdienst zum Schicken bzw. Empfangen der Nachrichten, einem Authentication-Protokoll zur Beglaubigung gesendeter Nachrichten zu schützen, und einem Privacy-Protokoll zur Entschlüsselung empfangener Nachrichten besteht.

In SNMPv2 gibt es außer der 5 Operationen von SNMPv1 noch drei neue:

1. `GetBulkRequest`: Durch diesen Befehl kann ein Manager eine große Menge von Daten, wie alle Einträge einer Tabelle, aus einem Agenten lesen.

2. `InformRequest`: Diese Operation erlaubt einem Manager, Verwaltungsinformationen zu einem anderen zu schicken.

3. `SNMPv2-Trap`: Die alte Trap-Meldung von SNMPv1 wurde in der neuen Version durch diese vereinfachte Trap-Meldung ersetzt. Die neue Trap-Meldung besitzt im Gegensatz zu seinem Vorgänger die selbe PDU-Struktur wie alle andere Operationen.

2.10 SNMPv3

Für viele Fälle waren die Sicherheitskriterien von SNMPv2 noch immer unausreichend, obwohl diese Version gegenüber seinem Vorgänger viele Erweiterungen im Sinne der Sicherheit beinhaltet. Um dieses Problem zu beheben, hat man eine dritte Version (SNMPv3) mit vielen neuen Sicherheitsmaßnahmen herausgegeben. Außer den neuen Sicherheitstasks stellt SNMPv3 neue Erweiterungen im Rahmen der Administration zur Verfügung, wobei der Rest der Architektur von SNMPv2 übernommen wurde.

SNMPv3 erweitert den herkömmlichen SNMP-PDU um einen weiteren Teil, der von einem Model für die Verarbeitung und den Versand von Nachrichten, nämlich "Message Processing and Dispatching" (MPD), verwendet wird. MPD ist ein universelles Model für die Verwaltung von Nachrichten und PDUs. Es ist für das Empfangen und Schicken von PDUs der drei SNMP-Versionen zuständig. Es verkapselt die empfangenen PDUs in die Nachrichten und ruft das sogenannte User Security Model (USM) [RFC2274] auf, welches ein Sicherheitssystem mit verschiedenen Authentifizierungs- und Datenschutzdiensten ist, mit denen unerlaubte Modifikationen von Nachrichten und Verbindungen, Verfälschungen von Authentifizierungen und Mithören der Kommunikation vermieden werden sollen. Es fügt in die Nachrichten sicherheitsbezogene Parameter ein. Wenn Nachrichten verschickt werden sollen, verarbeitet das USM die aus den Nachrichten-Header erhaltenen Parameter. Dann packt das MPD die Nachrichten in die entsprechende PDUs ein und verschickt sie zu ihren Empfängern.

Außer USM wurde ein weiteres Sicherheitkonzept namens View-Based Access Control (VBAC) [RFC2275] in die dritte Version von SNMP integriert. VBAC bestimmt die Rechte für Fernzugriffe auf verwalteten Objekte in einer SNMP-MIB. Dabei benutzt VBAC ein eigenes MIB, die die Zugriffsrechte für den jeweiligen Knoten beinhaltet. Diese MIB kann fernkonfiguriert werden. Außerdem ermöglicht VBAC das, daß mehrere Mechanismen für die Zugriffssteuerung für einen einzigen Knoten implementiert werden, obwohl dieser Fall in der Praxis nicht viel zu finden ist.

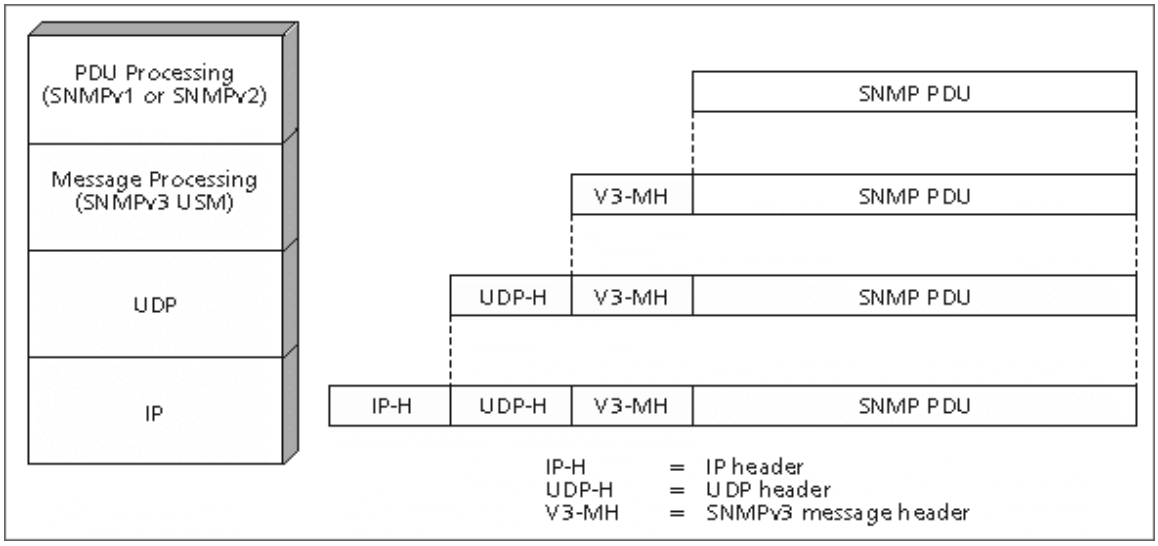


Abb. 2.15 Zusammenhang zwischen PDUs verschiedener SNMP-Versionen [STA98]

3. DAS FELDBUSSYSTEM P-NET

3.1 Die Grundprinzipien von P-NET

P-NET ist ein Feldbussystem, das von der dänischen Firma Proces-Data A/S entwickelt wurde, um verteilte Prozeßkomponenten wie Prozeßcomputer, intelligente Sensoren und Aktoren, SPS-Controller usw. über eine gemeinsame Zweidrahtleitung, siehe Abb. 3.1, zu verbinden und dadurch die konventionelle Verdrahtung mit ihrem sehr hohen Verdrahtungsaufwand zu ersetzen.

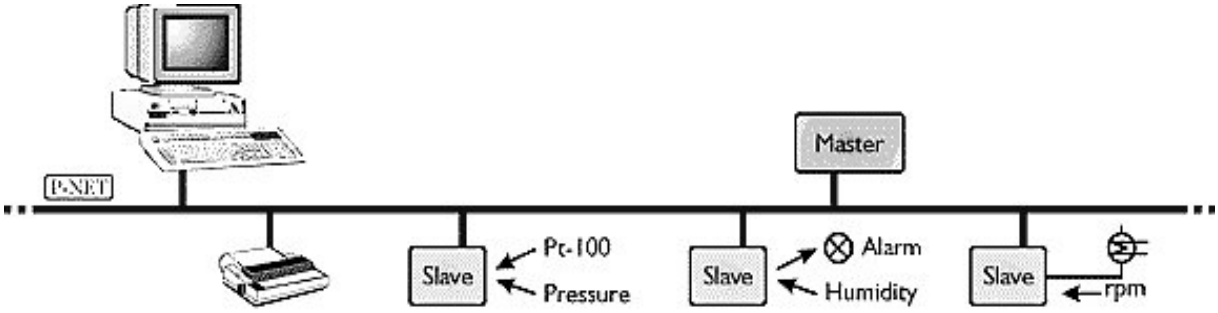


Abb. 3.1 P-NET mit verteilten Prozeßkomponenten [PNET96]

P-NET wurde im Jahre 1983 entwickelt und wurde 1996 wie zwei andere Feldbussysteme, PROFIBUS und WorldFIP, mit der europäischen Norm EN50170 standardisiert.

P-NET ist ein mächtiges, aber auch flexibles und universell einsetzbares Feldbussystem, das einen physikalischen Bus, der zu einem Ring zusammengeschlossen ist, als Übertragungsmedium verwendet (passiver Ring). Innerhalb eines Bussegments können bis zu 125 Geräte ohne Einsatz von Repeatern angeschlossen werden, wobei maximal 32 davon Master-Geräte sein können. Ein Master darf Nachrichten ohne externe Aufforderung aussenden, wenn er die Buszugriffsberechtigung (Token) besitzt. P-NET verwendet dabei eine "virtual token passing" genannte Methode. Diese Methode wird im Punkt 3.3.2 ausführlich erläutert. Durch dieses Tokenverfahren wurde die Echtzeitfähigkeit des Kommunikationsprotokolls gewährleistet. Slave-Geräte sind Peripheriegeräte, die den Bus nur nach der Aufforderung eines Masters zum Senden einer Antwort benutzen dürfen.

Neben der Multimasterfähigkeit können auch Bussegmente unter der Verwendung der Multiport-Master zu richtigen Netzwerken verbunden werden, was auch als "Multi-Net-Struktur" bekannt ist. Da man durch diese Struktur die Programmausführung und die dafür

nötige Intelligenz auf ein oder mehrere Teilnetze verteilen kann, können redundante und fehlertolerante Systeme aufgebaut werden. Außerdem wird damit der starke Datenverkehr zwischen den Knoten reduziert. Eine solche Multi-Net-Struktur wird in der Abbildung 3.2 dargestellt.

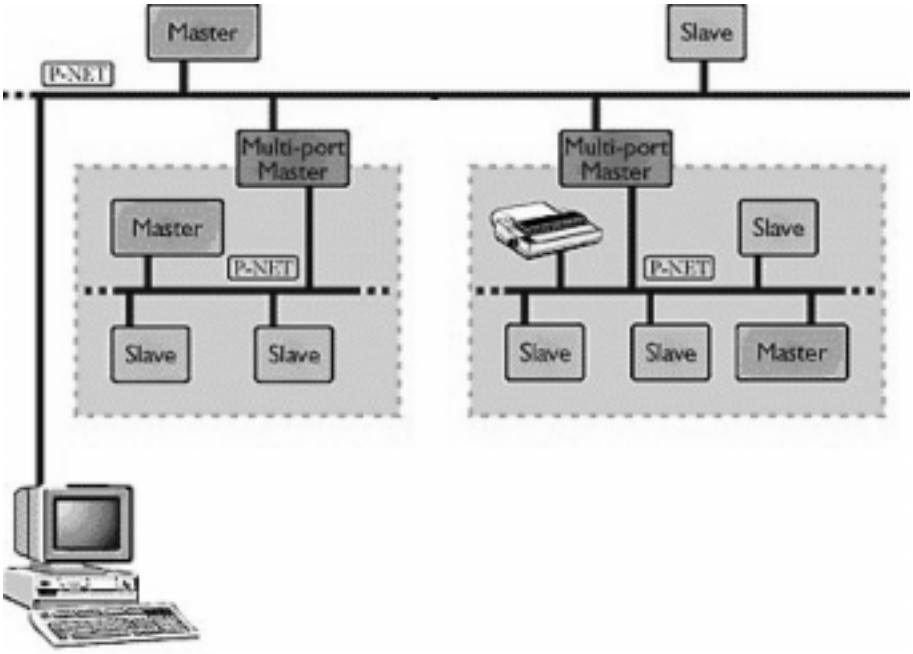


Abb. 3.2 Eine Multi-Net-Struktur mit P-NET [PNET96]

Die Interoperabilität ist einer der Vorteile von P-NET, wobei diese durch ein einschränkendes Protokoll, ähnlich wie bei Interbus-S, gewährleistet wird. Jedes P-NET Modul beinhaltet mehrere Channels, die eine objektartige Gruppierung von mehreren Werten wie Ein- und Ausgangswerte, Grenzwerte usw. sind. Da die meisten Channels standardisiert sind, bleiben die Kompatibilitätsprobleme zwischen den Geräten verschiedener Hersteller erspart. So können die Module sehr einfach miteinander kommunizieren und problemlos mit den Modulen anderer Hersteller ausgetauscht werden.

Gegenüber einfachen Single-Master-Systemen wie Interbus-S und ASI, ist P-NET sehr langsam. Die typische Antwortzeiten liegen im Millisekunden-Bereich. Für Applikationen, bei denen die Antwortzeit einige Mikrosekunden dauern soll, ist P-NET nicht geeignet.

Als Programmiersprache speziell für die Controller, die intelligenten Geräte am Bus, wird Process-Pascal benutzt, welches das um Netzwerkvariablen und Multitaskingfähigkeit erweiterte Standard-ISO-Pascal ist. Im Gegensatz zu vielen anderen Feldbussystemen verfügen die "unintelligenten" Slave-Einheiten außer herkömmlichen Ein- und Ausgabefähigkeiten meistens auch über zusätzliche prozeßnahe Funktionen, wie z.B. die Umrechnung von Meßwerten oder eine PID-Regelung, und können in einer einfachen Assemblersprache namens "Calculator Assembler" programmiert werden.

Für die Ankopplung von P-NET mit PCs wurde VIGO entwickelt, das die Objekte am P-NET als OLE2-Objekte zur Verfügung stellt, so daß sie von anderen Windows Applikationen verwendet werden können.

Trotz seiner Vorteile ist P-NET sehr wenig verbreitet. P-NET-Applikationen werden dort verwendet, wo analoge Prozeßdaten erfaßt und ausgewertet werden müssen, beispielweise in der Verfahrenstechnik, Lebensmittelindustrie und Umwelttechnik.

3.2 Die Architektur von P-NET

Im P-NET sind die Schichten 1, 2, 3, 4 und 7 des OSI-Referenzmodells implementiert.

3.2.1 Die Bitübertragungsschicht

Als Übertragungstechnik wird bei P-NET der EIA RS-485 Standard eingesetzt, wobei das Kabel zu einem physikalischen Ring geschlossen wird, um die in diesem Standard vorgeschriebenen Abschlußwiderstände zu vermeiden. Dadurch ist auch eine höhere Anzahl von angeschlossenen Knoten als die 32 im RS-485 Standard erlaubten zugelassen.

Es können maximal 125 Knoten angeschlossen werden, weil für die Adressierung nur 7 Bits vorgesehen sind und dadurch nur die Adressen zwischen 0 und 127 zur Verfügung stehen, wobei 3 Adressen davon für bestimmte Zwecke reserviert sind.

Zusammenfassend hat P-NET in dieser Schicht folgende Merkmale:

- Leitung : Twisted Pair + Shield
- max. Buslänge: 1200m
- max. Knotenanzahl: 125
- Übertragungsrate: 76800 Baud

Da P-NET nur eine Datenrate verwendet, können Geräte direkt an den Bus gekoppelt werden und können sofort miteinander kommunizieren, während bei vielen anderen Systemen die unterschiedlichen Datenraten zu Problemen der Kommunikation untereinander führen können.

3.2.2 Die Sicherungsschicht

Da P-NET multimasterfähig ist und bis zu 32 Master-Geräte in einem Bussegment erlaubt, müssen die Zugriffsrechte auf den Bus verwaltet werden. In dieser Schicht wird das durch dieses Virtual-Token-Passing-Verfahren verwirklicht.

Jeder Master-Knoten hat eine Knotenadresse (Node Address) zwischen 1 und der erwarteten Gesamtanzahl des Masters (max. 32) und besitzt außer einem Leerlaufzähler (Idle Bus Bit Period Counter), der mit jeder Bitperiode inkrementiert wird, in der der Bus inaktiv (idle) ist, noch einen Zugriffszähler, der um 1 erhöht wird, wenn der Leerlaufzähler 40, 50, 60, ... erreicht.

Ein Master erhält die Buszugriffsberechtigung, also den Token, wenn sein Zugriffszähler den Wert seiner Knotenadresse erreicht (siehe Abb. 3.3). Wenn der Zugriffszähler die Gesamtanzahl der Master überschreitet, wird er wieder auf 1 zurückgesetzt.

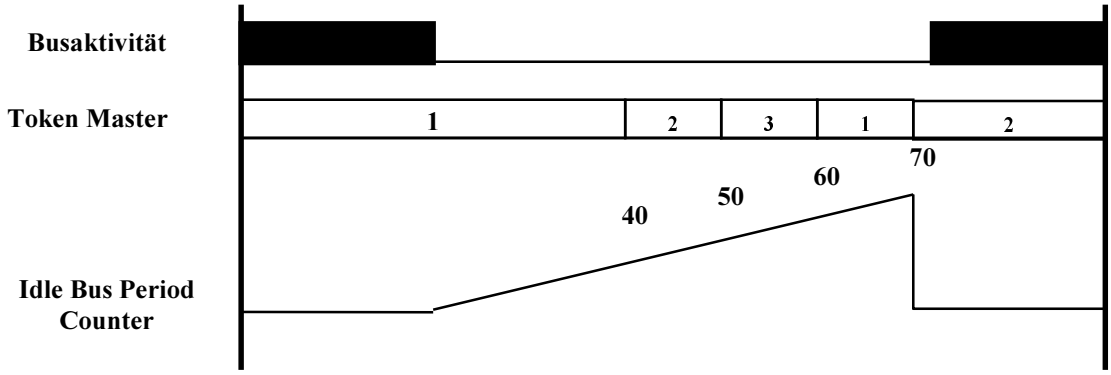


Abb. 3.3 Virtual Token Passing mit 3 Master.

In diesem Beispiel hat am Anfang der Master 1 den Token und benutzt den Bus. Daraufhin wird der Bus inaktiv und der Leerlaufzähler wird laufend inkrementiert. Sobald dieser den Wert 40 erreicht, werden alle Zugriffszähler um 1 erhöht und der Token wird virtuell an Master 2 weitergegeben. Und immer wenn in den nächsten Zeitspannen von 10 Bitperioden der Bus von dem jeweiligen Master nicht benutzt wird, wird der Token immer dem nächsten Master übergeben. So geht der Token beim Stand von 50 über zu Master 3, nach weiteren 10 Bitperioden weiter zu Master 1 und dann wieder zu Master 2. Dieser hat in der Zwischenzeit Verwendung für den Bus und kann ihn für genau eine Anfrage benutzen. [MM96]

Slaves dürfen nur in einem Zeitintervall zwischen 11 und 30 Bitperioden auf den Bus nach einer Anfrage von einem Master zugreifen. Dadurch ist die maximale Zeit zwischen einer Anfrage und Antwort auf 390µs beschränkt.

In der Sicherungsschicht werden die zu sendenden Daten in ein Datenpaket (Frame) verpackt und die Absender- und Zieladressen in das Adreßfeld dieses Pakets eingefügt, wobei der Adreßtyp einfach oder komplex sein kann (siehe Abb. 3.4). Beim komplexen Adreßtyp gibt das dritte Byte die Anzahl der nachfolgend übertragenen Adreßbytes an. Auf diese Weise ist eine Adressierung über mehrere P-NET–Ports hinweg möglich. Der erweiterte Adreßtyp ist ein Spezialfall des komplexen Adreßtyps, der bei Anforderungen verwendet wird. Es handelt sich um nur zwei Ziel- und Quellenadressen.

3.2.3 Die Vermittlungsschicht

Im P-NET können Bussegmente durch Master mit zwei Ports, sogenannte Multiport-Master, miteinander zu einem Netzwerk verbunden werden, so daß eine verteilte Intelligenz zwischen den Zell-Controllern, den Interfaces und den Sensoren möglich ist. Das ermöglicht auf den höheren Ebenen eine Reduktion der nötigen Datenrate. Außerdem kann man dadurch redundante und fehlertolerante Systeme aufbauen.

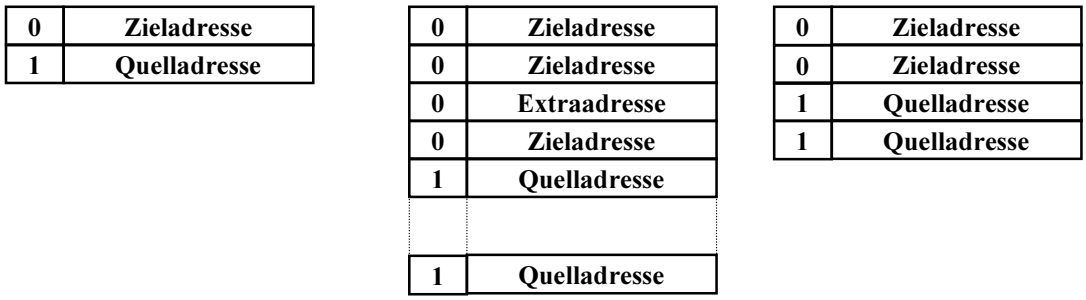


Abb. 3.4 Einfache, komplexe und erweiterte Adresstypen

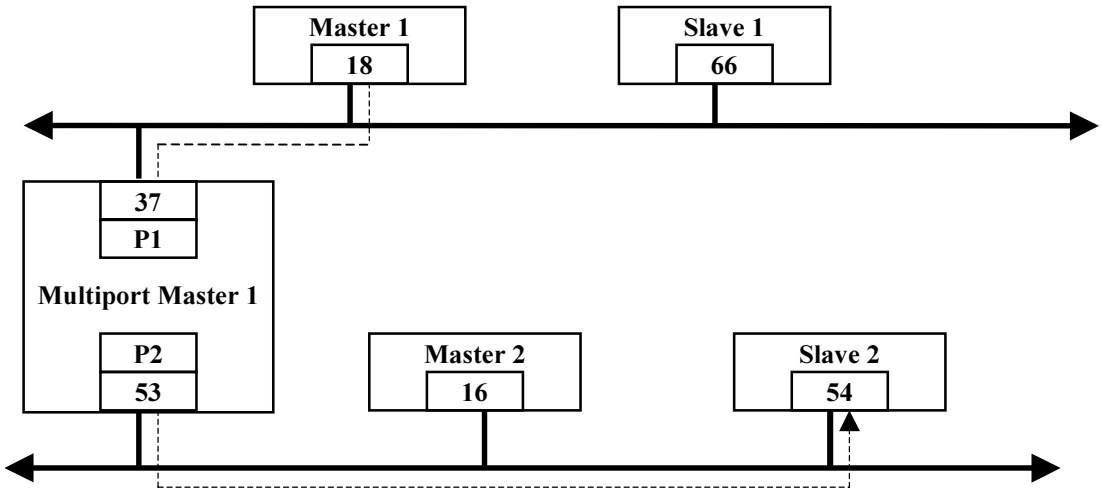


Abb. 3.5 Routing im P-NET

Da die Vermittlungsschicht die Funktion eines Routers zwischen zwei Bussegmenten übernimmt, ist eine ihrer Aufgaben das Routing des Übertragungsweges. Jeder Master eines Bussegments kann durch dieses Routingverfahren transparent auf jeden Knoten innerhalb eines anderen Bussegments zugreifen. Dieses Verfahren wird durch ein Beispiel in der Abbildung 3.5 dargestellt. Wenn ein Master eine Anfrage an einen Slave in einem anderen Segment richtet, stehen vorerst im Adreßfeld eine Reihe von Zieladreßbytes und die Quellenadresse des Masters selbst. Um in ein anderes Segment zu gelangen, ist die externe Adresse des Multiport-Masters, der die zwei Segmente miteinander verbindet, erforderlich. Beim Erreichen dieses Multiport-Masters wird diese externe Adresse durch eine interne "Quellenadresse" ersetzt, die die interne Portnummer beschreibt. Beim Verlassen dieses Masters wird die nächste Zieladresse, die die Ausgangsportnummer angibt, durch die externe Adresse an diesem Port ersetzt. Diese Adressierungsumwandlungen für das in der Abbildung 4.5 gezeigtes Beispiel werden in der Abbildung 3.6 beschrieben.

Der Vorteil dieses Verfahrens ist, daß der Slave dieses umgewandelten Adreßfeldes für das Zurücksenden seiner Antwort an den jeweiligen Master verwenden kann.

Der Nachteil ist, daß bei einer Änderung in der Netzstruktur auch alle Programme geändert werden sollen, so daß sie den neuen Weg zu den entfernten Knoten wieder finden können.

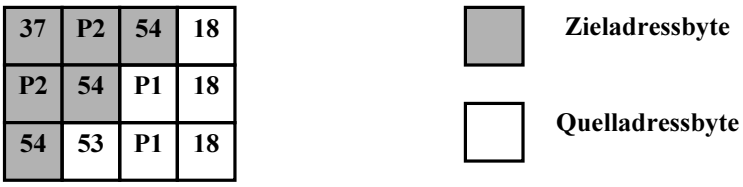


Abb. 3.6 Adressfeldumwandlung bei einer Anfrage

3.2.4 Die Transportschicht

Bei P-NET werden alle globalen Variablen wie Knotenadressen, Meßwerte, Variablentypen usw. auf einer Liste namens Softwirelist angeführt. Die Variablen werden durch ihre logische Adresse, die sogenannten Softwire Number (SWNo), zur Verfügung gestellt. Falls eine Variable physikalisch einem externen Knoten zugeordnet ist, dann wird nur ihre Knotenadresse, wo die eigentliche Adresse der Variable zu finden ist, in die Softwirelist eingetragen. Durch diese Adresse kann die Variable auf dem zugehörigen Knoten zugegriffen werden. Diese Schicht übernimmt die Aufgabe, Daten in die Softwirelists zu schreiben bzw. aus diesen Listen zu lesen. Außerdem besitzt diese Schicht eine zweite Task, die alle Details über alle Anfragen sammelt, die ausgeschickt wurden, aber noch auf eine Antwort warten. Nach der Ankunft der Antwort wird sie zur anfordernden Applikation zurückgesandt.

3.2.5 Die Anwendungsschicht

Im P-NET greifen die Anwendungsprogramme auf Variablen mittels ihrer SWNo zu, wobei die Variablen auf dem selben Knoten wie das Programm selbst oder auf einem anderen Knoten sein können. Im zweiten Fall steht in der Softwirelist die Adresse des Knotens, wo sich die Variable befindet. Die eigentliche Adresse dieser Variable steht in der Softwirelist dieses Knotens.

3.3 Die Channel-Struktur

Bei P-NET besitzen die meisten Geräte Prozeßsignale wie digitale Ausgänge oder analoge Eingänge, die aber auch zusätzliche Daten und Funktionen zur Konfiguration, Umwandlung, Skalierung usw. benötigen, um richtig verarbeitet werden zu können.

Alle auf ein Prozeßsignal bezogenen Variablen und Funktionen werden zu einem Objekt namens Channel zusammengefaßt. Ein Channel beinhaltet außerdem Daten für die Wartung und Datenübertragung. Als Beispiel wurde der digitale I/O-Channel in der Abb. 3.7 gezeigt.

SWNo	Identifier	Art des Speichers	Format	Type	Einheit
x0	FlagReg	RAM Read Write	binär	Array	
x1	OutTimer	RAM Read Write	dezimal	Real	s
x2	Counter	RAM Auto Save	dezimal	LongInt	
x3	OutCurrent	RAM Read Write	dezimal	Real	A
x4	Operatingtime	RAM Auto Save	dezimal	Real	s
x5					
x6	FPTimer	RAM Read Write	dezimal	Real	s
x7	FBPreset	EEPROM RPW	dezimal	Real	s
x8	OutPreset	EEPROM RPW	dezimal	Real	s
x9	ChConfig	EEPROM RPW		Record	
xA	MinCurrent	EEPROM RPW	dezimal	Real	A
xB	MaxCurrent	EEPROM RPW	dezimal	Real	A
xC					
xD	Maintenance	EEPROM RPW		Record	
xE	ChType	PROM Read Only		Record	
xF	CHError	RAM Read Write	binär	Record	

Abb. 3.7 Die Struktur des digitalen I/O-Channels

Ein Channel besteht aus 16 Registern, von denen jedes eine eigene logische Adresse, nämlich Softwire Number (SWNo), hat. Die Daten in diesen Registern können sowohl von einfachen Datentypen wie Byte, Integer, Boolean, String usw. als auch ein Rekord von diesen Typen sein. Sie können in ROM, RAM oder EEPROM gespeichert werden.

3.4 Process-Pascal

Bei P-NET wird Process-Pascal als Programmiersprache speziell für Controller eingesetzt. Diese Sprache basiert auf ISO7185-Standard-Pascal, der auf Multitaskingfähigkeit, zusätzliche Datentypen für Netzwerkvariablen (Buffer, Channel und Interface) und Funktionen zur Interaktion an einem LC-Display erweitert wurde.

Durch die Multitaskingfähigkeit von Process-Pascal können auf einer Controller-CPU mehrere Tasks und kleine Unterprogramme quasi parallel laufen. Dabei wird zwischen den Tasks umgeschaltet, das durch die Standardprozedur CHANGETASK erfolgt.

Bei P-NET gibt es drei Arten von Tasks:

1. Cyclic-Tasks, die zyklisch bearbeitet werden.

2. Timedinterrupted-Tasks, die in bestimmten Zeitintervallen aufgerufen werden.

3. Softwireinterrupted-Tasks, die bei Auftreten eines bestimmten Zustandes aufgerufen werden.

Die Softwire- und Timedinterrupted-Tasks haben höhere Priorität als Cyclic-Tasks. Falls eine Softwire- bzw. Timedinterrupted-Task sich während der Ausführung einer Cyclic-Task meldet, dann wird die Cyclic-Task unterbrochen und die neue Task wird bearbeitet. Erst nach der Beendigung der Interrupted-Task wird wieder die Durchführung der Cyclic-Task fortgesetzt. Die Abbildung 3.8 zeigt ein Beispiel für ein mögliches Multitaskingverfahren mit 4 Cyclic-Tasks und 2 Softwire- bzw. Timedinterrupted-Task.

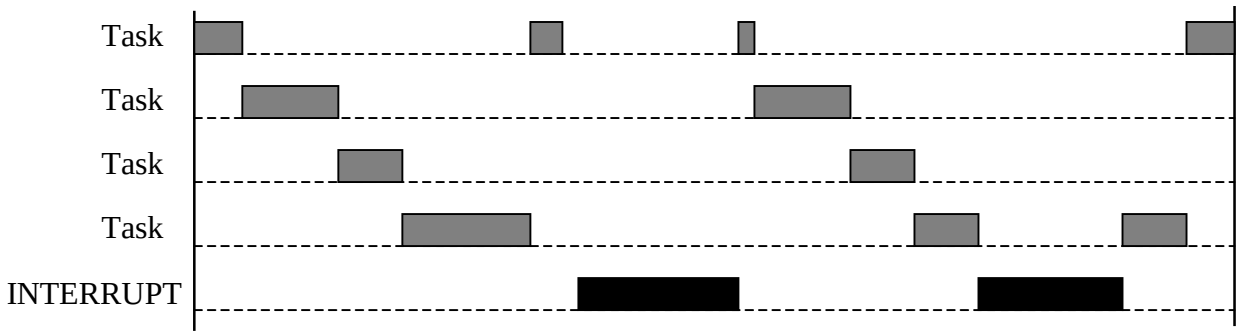


Abb. 3.8 Multitaskverfahren mit 4 Cyclic Tasks und 2 Softwire- bzw. Timedinterrupted Task

Bei Process Pascal werden drei Datentypen, `Buffer`, `Interface` und `Channel`, zusätzlich zu den Standardtypen von ISO-Pascal unterstützt, um Netzwerkvariablen deklarieren zu können. `Buffer` ist für die Variablen, die ein Zwischenspeicher referenzieren sollen, der nach dem First-In-First-Out (FIFO) Prinzip arbeitet. Der Datentyp `Interface` ist wie ein Record eine Sammlung von allen möglichen Datentypen, während jedem davon eine SWNo. zugeordnet ist und alle 16 SWNo.s in einem `Channel` zusammengefaßt sind.

Für ein Zugriff auf eine externe Variable ist im Process-Pascal Programm die Deklaration seiner physikalischen Adresse notwendig, die durch ein definiertes Interface Modul, die P-NET-Adresse und die Nummer des P-NET-Ports beispielweise folgendermaßen geschieht [PPM91]:

```
(*$I'PDMODULE.DEF*)

var
AnaModule : PD1611 AT NET: ( 1,$38);
```

Um eine Variable indirekt mit einem kürzeren Namen zu referenzieren, braucht man zum Beispiel folgende Zeile einzutragen:

```
Temperature -> AnaModule.Ch1.AnalogIn;
```

In weiteren Teilen des Programms kann man für den Zugriff auf die Variable statt dem langen Namen "AnaModule.Ch1.AnalogIn" einfach den verständlicheren "Temperatur" benutzen.

3.5 VIGO

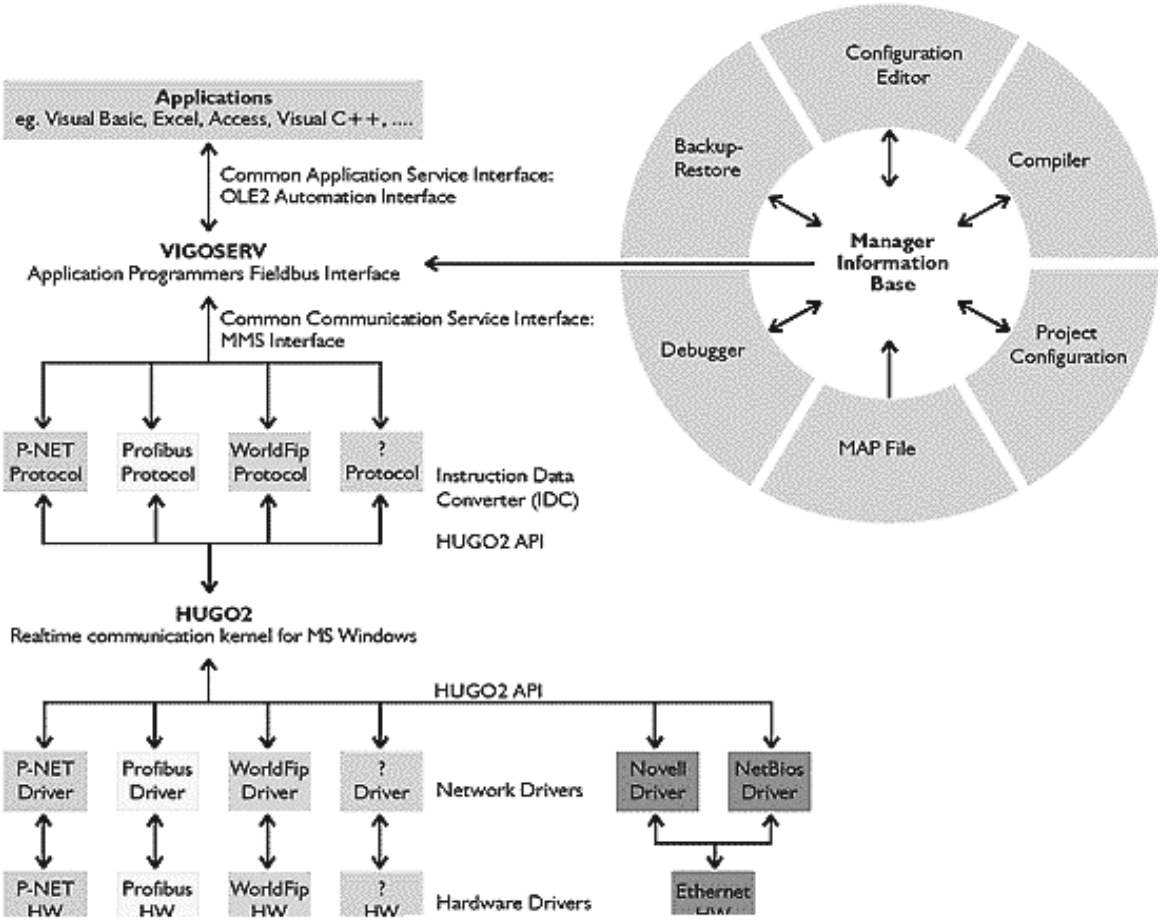


Abb. 3.9 Die Elemente von VIGO [VUM97]

Virtual Interface using Graphical Objects (VIGO) ist ein Feldbus-Management-System, welches auf den Computern mit MS Windows 95 oder NT als Betriebsystem laufen kann und mit Hilfe einer speziellen in den Computern eingebauten Karte den Zugriff auf P-NET ermöglicht.

VIGO ist eine standardisierte Benutzerschnittstelle, die in Wirklichkeit eine Ansammlung von diversen Programmelementen ist (siehe Abb. 3.9). Es ist ein offenes System, da das Integrieren neuer Netzwerkelemente anderer Hersteller ins VIGO möglich ist. Alle diese Elemente werden von VIGO verwaltet, so daß eine einfache und flexible Schnittstelle für etliche Feldbussysteme zur Verfügung gestellt werden kann.

3.5.1 VIGOSERV

Application Programmers Fieldbus Interface (VIGOSERV) ist ein OLE2 Automation-Server, der eine einheitliche und transparente Schnittstelle zwischen Anwendungsprogrammen und physikalischen Objekten ist und einen Echtzeit-Datenaustausch zwischen MS-Windows-Programmen erlaubt. Damit werden über VIGO die physikalischen Objekte allen Applikationen zur Verfügung gestellt, die die OLE (Object Linking & Embedding) unterstützen. Solche Applikationen sind zum Beispiel weit verbreitete kommerzielle Windows-Programme wie MS Excel bzw. MS Access oder mit objektorientierten Programmiersprachen wie Visual Basic, Delphi, C++ usw. geschriebene anwendungsspezifische Programme.

Durch VIGOSERV wird es den Anwendungsprogrammen ermöglicht, den Wert eines physikalischen Objekts zu lesen und zu verändern. Das physikalische Objekt soll nur über ein virtuelles Objekt in VIGO angesprochen werden, welches im Programm durch einen Virtual Identifier gekennzeichnet wird (siehe Abb. 3.10).

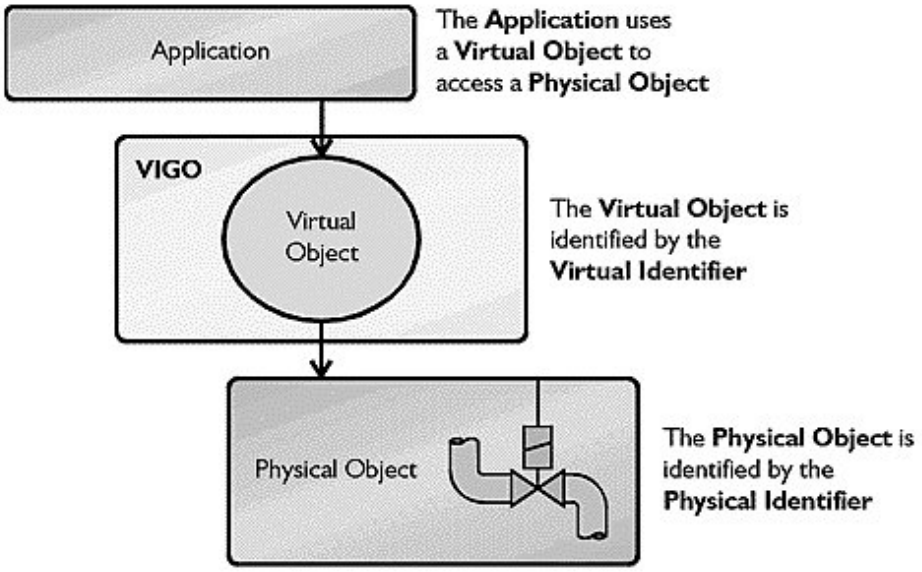


Abb. 3.10 Virtuelles und physikalisches VIGO-Objekt [VUM97]

Im folgenden ist ein in Delphi geschriebenes Beispiel aufgeführt, das aus vier Schritten besteht:

1. Schritt: Der Virtual Identifier des virtuellen Objektes wird deklariert, mit dem man auf ein P-NET-Objekt zugreifen wird.

```
Gewicht : variant;
```

2. Schritt: Mit der folgenden Zeile wird das Objekt erzeugt. Man kann in verschiedenen Programmen das selbe physikalische Objekt, sogar auch mit dem selben Identifier, ansprechen.

```
Gewicht := createoleobject('VIGO');
```

3. Schritt: Die physikalische Adresse wird dem virtuellen Objekt zugeordnet.

```
Gewicht.PhysID := 'Project1:Waage.WEIGHT.Weight0'
```

Nach diesen Schritten kann die Variable "Gewicht" in dem Programm wie eine normale Variable verwendet werden, so daß ihr Wert und somit der Zustand des physikalischen Objekts gelesen und geändert wird.

4. Schritt: Auf das physikalische Objekt wird zugegriffen.

Es gibt zwei Zugriffsmöglichkeiten:

- Bei der ersten Methode hält die Applikation nach einem Zugriffsversuch solange an, bis der Wert des physikalischen Objekts geholt ist. Diese Methode ist als "externer Zugriff" bzw. "direkter Zugriff" bezeichnet.

- Bei der zweiten Methode, die "interner Zugriff" bzw. "gepufferter Zugriff" heißt, fährt das Programm fort, ohne auf den Wert zu warten. Der Wert wird dann in einem Puffer in VIGOSERV gespeichert, aus dem das Programm ihn später lesen kann. Diese Methode ermöglicht parallele Programmbearbeitung. Wenn mehrere Werte gleichzeitig abgefragt werden, kann das Programm mit denen weiter arbeiten, die schnell zu empfangen sind, bis es auch die anderen Werte bekommen kann.

Für den externen Zugriff muß man der Typeneigenschaft die Vorsilbe "Ex" bzw. für den internen Zugriff "In" einsetzen. Für unser Beispiel sieht das folgendermaßen aus:

1. Externer Zugriff:

   - Lesen des Wertes:

     ```
     X := Gewicht.ExValue;
     ```

   - Ändern des Wertes:

     ```
     Gewicht.ExValue := 23.4;
     ```

2. Interner Zugriff:

   - Lesen des Wertes:

     ```
     Gewicht.DoRead;          (Anforderung für das Lesen des Wertes)
     X = Gewicht.InValue;     (Lesen des Wertes)
     ```

- Ändern des Wertes:

```
Gewicht.InValue := 23.4;
Gewicht.DoWrite;            (Schreiben des Wertes)
```

Falls der Wert der physikalischen Variable noch nicht erreicht ist, d.h. bis die Applikation ihn tatsächlich braucht, dann wird sie wie beim externen Zugriff solange angehalten, bis der Wert erlangt werden kann.

3.5.2 Manager Information Base (MIB)

Manager Information Base ist eine Datenbank, in der die Daten über die Struktur des physikalischen Systems einer Anlage, in VIGO als Projekt bezeichnet, gespeichert sind.

MIB beinhaltet alle notwendigen Informationen für den Zugriff auf eine Variable in einem Knoten und übersetzt einen Variablennamen in einer Applikation in eine spezifische Knotenadresse, Variablenadresse innerhalb des jeweiligen Knotens, Typenspezifikation, usw.

Objekte in der MIB können einfache Datentypen, wie Boolean, Byte, Integer, Real, usw. oder komplexe, wie Array, Record oder String, haben. Alle Variablen eines Knotens werden in einer Record-Variable, gekennzeichnet mit einem Node Identifier, gesammelt, wobei alle Knoten in einer weiteren Record-Variable gespeichert werden, die durch einen sogenannten Project Identifier angesprochen werden kann. Um auf eine Variable zuzugreifen, wird ein Global Identifier benötigt, der aus dem Project Identifier, Node Identifier und Variable Identifier, mit dem die Variable innerhalb des Knoten identifiziert werden kann, folgendermaßen gebildet wird:

```
Project_Identifier:Node_Identifier.Variable_Identifier
```

Wenn VIGOSERV diesen Global Identifier zur MIB schickt, um auf ein physikalisches Objekt zuzugreifen, sammelt die MIB alle benötigten Informationen über dieses Objekt und sendet sie zum VIGOSERV zurück.

Das VIGO-Programmpaket beinhaltet verschiedene Tools, mit denen man die MIB verwalten kann. Eines davon ist MIB Editor, welcher alle MIB-Daten in einer Baumstruktur darstellt (siehe Abb. 3.11). Da er ein *Microsoft Windows OLE Control Extension* (OCX) ist, kann er auch in andere Programme, wie Excel, Visual Basic, Visual C++ usw. integriert werden.

Mit diesem MIB-Editor kann man ein Gerät auswählen, dessen Daten anschauen und die Werte manuell ändern.

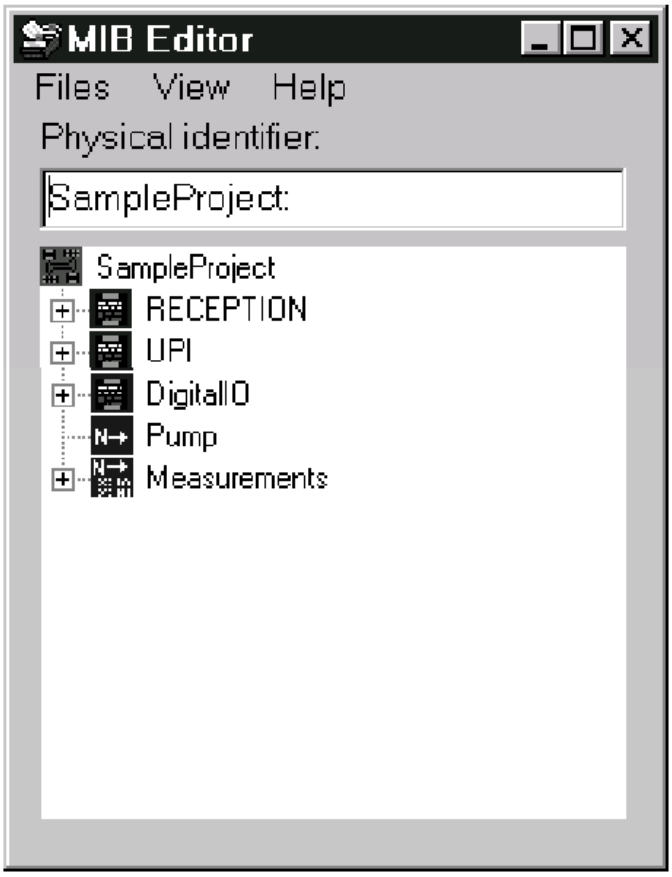


Abb. 3.11 MIB Editor [VUM97]

3.5.3 Instruction Data Converter (IDC)

VIGO wurde als ein einheitliches System für diverse Bussysteme entwickelt. Da aber unterschiedliche Bussysteme verschiedenartige Datenformate, Syntax und Dienste verwenden, braucht VIGO eine allgemeine Schnittstelle zwischen Applikationen und Feldbussystemen, mit der die Applikationen mit unterschiedlichen Feldbussen unabhängig von ihren spezifischen Datenformaten und Diensten kommunizieren können. Diese Schnittstelle wird von VIGOSERV zur Verfügung gestellt und heißt Common Communication Service Interface (siehe Abb. 3.12).

Der IDC wandelt eine VIGO-Anfrage in die jeweilige Feldbusanweisung um und schickt sie in dem für den Zielknoten erforderlichen Format weiter. Analog dazu, übersetzt IDC die aus der Feldbusschicht kommenden Daten in die Common Communication Service Interface Syntax.

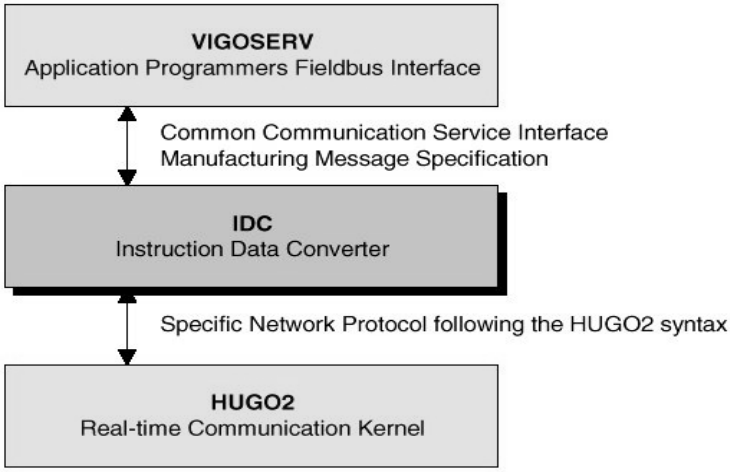


Abb. 3.12 Die Aufgabe eines IDCs [VUM97]

3.5.4 HUGO2

HUGO2 ist ein Echtzeit-Kommunikationskern, der für die parallele Ausführung verschiedener Kommunikationsprogramme zuständig ist. Er verwaltet die Buszugriffe und vermeidet Kollisionen, wenn mehrere Applikationen gleichzeitig auf den Bus zugreifen wollen.

HUGO2 unterstützt sowohl die zeitkritische als auch die nicht zeitkritische Kommunikation. Die zeitkritische Kommunikation wird mit den Interrupts gesteuert, während die nicht zeitkritische mit den Windows-Messages durchgeführt wird.[VUM97]

3.5.5 Network Drivers

Man kann an HUGO2 Feldbussysteme, Local Area Networks (LAN) und Wide Area Networks (WAN) anschließen. Das wird durch die Network Drivers ermöglicht, die für das Senden und Empfangen von Informationen über spezifische Netzwerke verantwortlich sind.

4. EINBETTUNG VON FELDBUSSYSTEMEN INS NETZMANAGEMENT

4.1 Einführung

Um die Ereignisse auf einem Feldbus zu protokollieren und eventuell auch zu visualisieren, genügt ein einfacher Personal Computer mit einer dem Feldbus entsprechenden Einsteckkarte, den benötigten Treibern und einer Standardsoftware. Bei P-NET können zum Beispiel die Feldbus-Daten über VIGO zugegriffen werden. Sie werden, wie in der letzten Kapitel genauer beschrieben, als Windows-OLE2-Objekte zur Verfügung gestellt, damit sie von gängigen Windows-Programmen, wie Excel oder Access, aber auch selbst geschriebenen Programmen angesprochen werden. Aber dadurch beschränkt man sich auf einen einzigen Computer. Die Feldbusdaten können primär nur auf diesem Rechner verarbeitet werden.

Durch einen direkten Zugriff auf den Feldbus aus einem übergeordneten Netz können Daten über größere Entfernungen hinweg erfaßt und bearbeitet werden. Diese Ankopplung des Feldbussystems an klassische LANs und damit an das Internet bietet mehrere Möglichkeiten, wie zum Beispiel zentrale Betriebsdatenerfassung, Fernüberwachung oder Fernwartung. Gegenüber klassische LANs hat das Internet dabei einige Vorteile, wie die Benutzung der standardisierten Internet-Protokolle, Verwendung einer bestehenden Infrastruktur, geringe Kosten für die Netzschaltung.

SNMP bietet dabei als ein erprobtes, einfaches aber trotzdem mächtiges Internet-Protokoll eine gute Möglichkeit für die Kommunikation zwischen den Netzmanagement-Stationen (NMS) und den an dem Feldbus angebundenen Agenten.

In dem Projekt "WEBFAN" am Institut für Computertechnik der Technischen Universität Wien wurde versucht, ein flexibles Netzmanagementsystem zu realisieren, mit dem auf verschiedene Feldbussysteme wie Profibus oder P-NET, über Internet unter Benutzung von SNMP zugegriffen werden kann. Diese Diplomarbeit beschäftigt sich mit einem Teil dieses Projekts, nämlich mit der Anbindung des Feldbussystems P-NET an das Netzmanagement.

4.2 Kopplungsmöglichkeiten zwischen LAN und Feldbus

Die Kopplung zweier Netzwerke kann durch verschiedene Koppeleinheiten erfolgt werden. Diese Koppeleinheiten sind Repeater, Bridges, Router und Gateways.

Repeater, die nur aus der ersten Schicht des OSI-Modells bestehen, verbinden zwei LANs miteinander, bei denen alle 7 Schichten gleich sind (siehe Abb 4.1). Ein Repeater ist eigentlich nur ein bitweiser Verstärker, der das Signal zwischen zwei gleichartigen Netzwerken verstärkt. Diese sind sehr kostengünstige und schnelle Koppeleinheiten, besitzen aber keine Intelligenz. Deswegen für eine Verbindung zwischen LAN und einem Feldbussystem, soll das Protokoll des Feldbusses in beiden Knoten der Verbindung implementiert werden. Das aber führt zu einer Inflexibilität der Verbindung, da bei jeder Änderung des Feldbusprotokolls alle Knoten auch aktualisiert werden müssen.

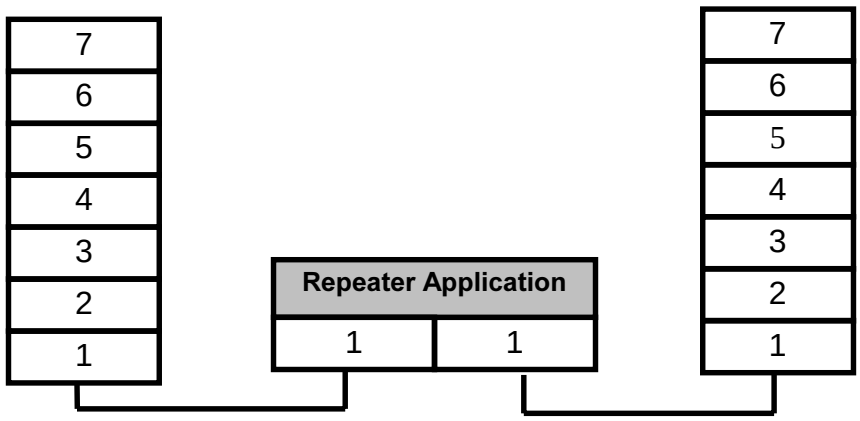


Abb. 4.1 OSI-Schichten einer Repeater-Kopplung

Bridges bestehen aus den ersten zwei Schichten des OSI-Modells und verbinden zwei Netzwerke miteinander, die die gleichen Schichten 3-7 besitzen (siehe Abb. 4.2).

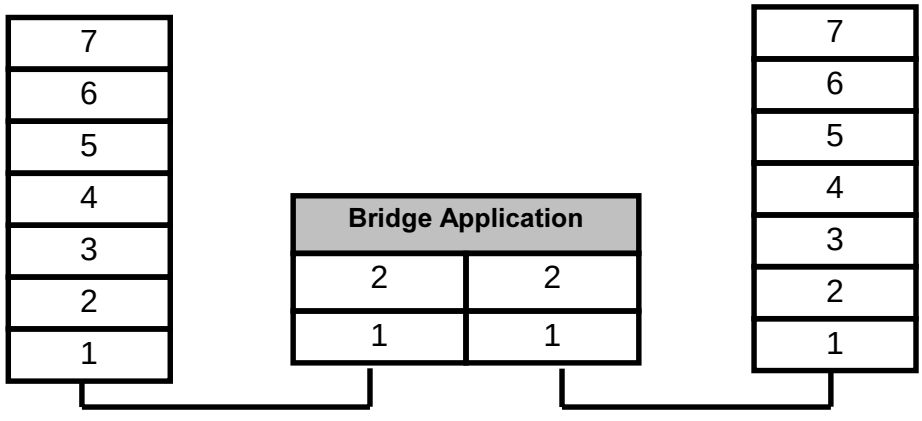


Abb. 4.2 OSI-Schichten einer .Bridge-Kopplung

Router verbinden Netzwerke mit der gleichen Schichten 4-7 miteinander. Sie erledigen auch die Aufgabe des Routings mithilfe von eigenen Algorithmen. Im Vergleich zu Bridges und Repeater, besitzen sie mehr Intelligenz und sind deshalb für eine Kopplung zwischen LAN und einem Feldbussystem geeigneter, obwohl sie auch teurer und langsamer sind. Da aber

diese Koppeleinheiten nur die ersten drei OSI-Schichten unterstützen, soll entweder das SNMP-Protokoll oder das Protokoll des Feldbusses in beiden Knoten der Verbindung implementiert werden. Die erste Variante erfordert intelligente Feldbus-Agenten, die die Protokollumsetzung übernehmen müssen (siehe Abb. 4.3). Da einfache Feldbus-Agenten mit dieser Aufgabe überlastet werden, ist diese Variante nicht geeignet für eine Feldbus-LAN-Verbindung.

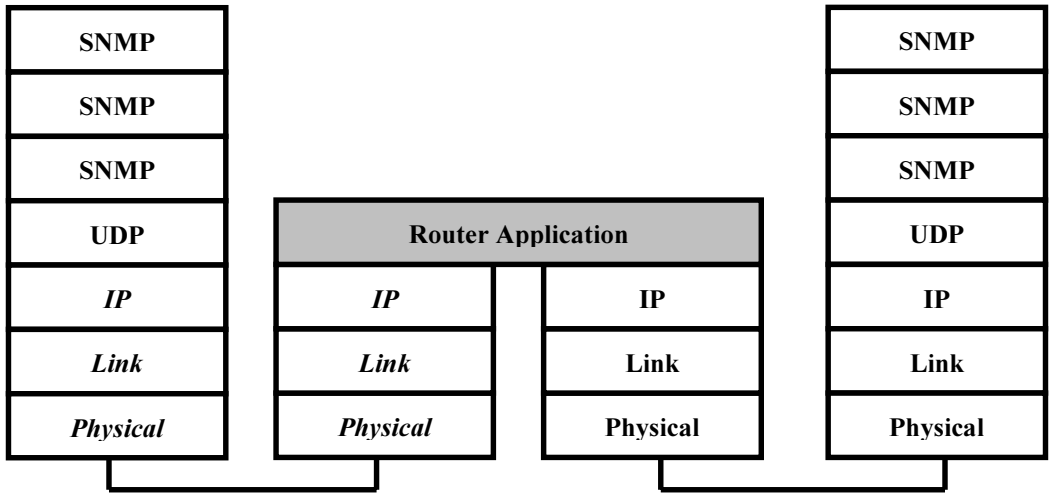


Abb. 4.3 Router mit SNMP-Protokoll

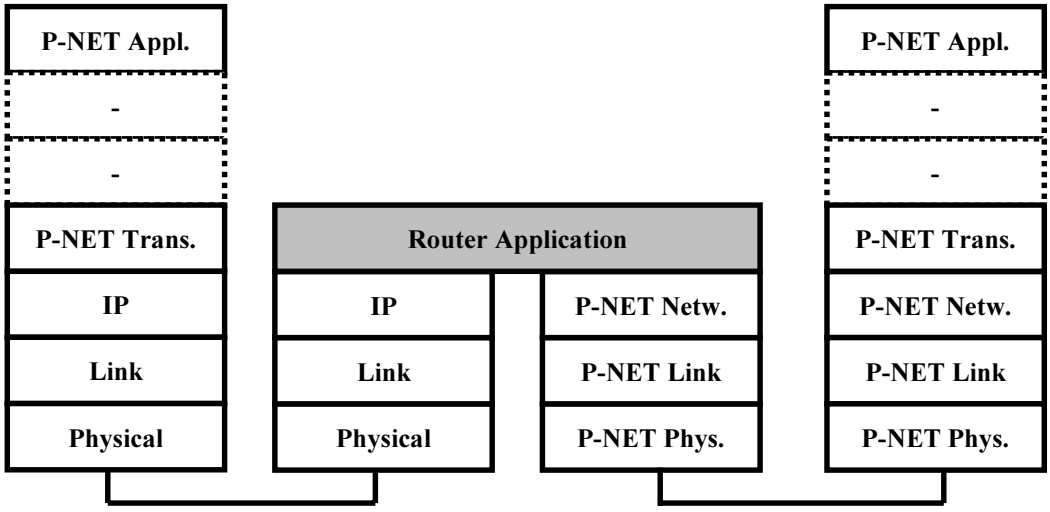


Abb. 4.4 Router mit P-NET Protokoll

Wenn man das Feldbus-Protokoll auf den beiden Seiten des Gateways implementiert, führt das zu einer Inflexibilität der Kopplung, da bei jeder Änderung des Feldbusprotokolls alle Knoten auch aktualisiert werden müssen und der Benutzer an dem anderen Ende der Kopplung auch detailliertes Wissen über den Feldbus besitzen muß. Deshalb ist diese

Kopplungsart, die einfach und billig zu implementieren ist, nur geeignet für Verbindungen, bei denen nur wenige Knoten auf den Feldbus zugreifen. In der Abbildung 4.4 ist eine Router-Verbindung mit P-NET Protokoll dargestellt.

Gateways sind Koppeleinheiten, die der Schicht 7 des OSI-Modells zugeordnet werden können. Sie verbinden Netzwerke miteinander, bei denen keine Schicht eines Netzwerks gleich der jenen des anderen ist. Sie beinhalten alle sieben Schichten des jeden Knotens. Die Abbildung 5.5 zeigt eine Kopplung von P-NET mit einem LAN via SNMP. Der Benutzer auf einer Seite der Kopplung braucht somit keine Kenntnisse über das Protokoll auf der anderen Seite zu besitzen. Eine Gateway-Applikation setzt die Datenpakete von einem Protokoll in die des anderen Protokolls um. Bei Änderungen an einem Protokoll, zum Beispiel auf der Feldbus-Seite, sollen entsprechende Modifikationen nur im Gateway vorgenommen werden. Ein weiterer Vorteil dieser Lösung ist die Flexibilität der Verbindung. Man kann zum Beispiel das Protokoll eines Feldbusses ganz einfach durch ein Protokoll eines anderen Feldbusses ersetzen, vorausgesetzt natürlich, daß das ganze Gateway modular aufgebaut ist. Obwohl die Komplexität und geringe Geschwindigkeit dieser Kopplung sich als Nachteile gegenüber anderen Koppeleinheiten erwiesen haben, wurde die Kopplung mit Gateway wegen den oben genannten Vorteilen als geeignetste Lösung für die Verbindung zwischen Feldbussystemen und LANs ausgewählt.

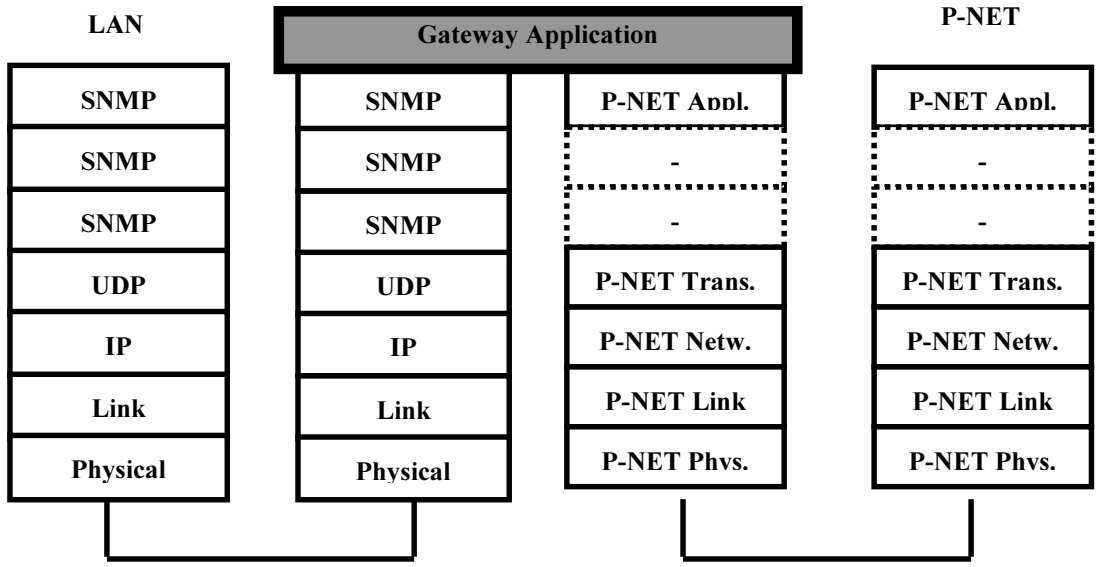


Abb. 4.5 Gateway zwischen LAN und P-NET

4.3 Struktur des Gateways

Heutzutage verwendet man eine Vielfalt von Feldbussystemen, von einfachen und schnellen Single-Master-Systemen wie Interbus-S oder PROFIBUS-DP bis zu mächtigen Multi-Master-

Systemen wie PROFIBUS-FMS oder P-NET. Um mehrere Feldbussysteme ans Internet anbinden und sie über Internet verwalten zu können, muß man im Prinzip für jedes Feldbussystem ein Gateway programmieren. Da kommuniziert die Netzmanagement-Station (NMS) mit je einem eigenen Agenten für jeden Feldbus, die die Aufgabe eines Gateways haben. (siehe Abb. 5.6) Das heißt aber, daß der ganze Agent für jeden neuen Feldbus neu implementiert werden muß, die wegen der Komplexität des Gateways sehr aufwendig werden kann.

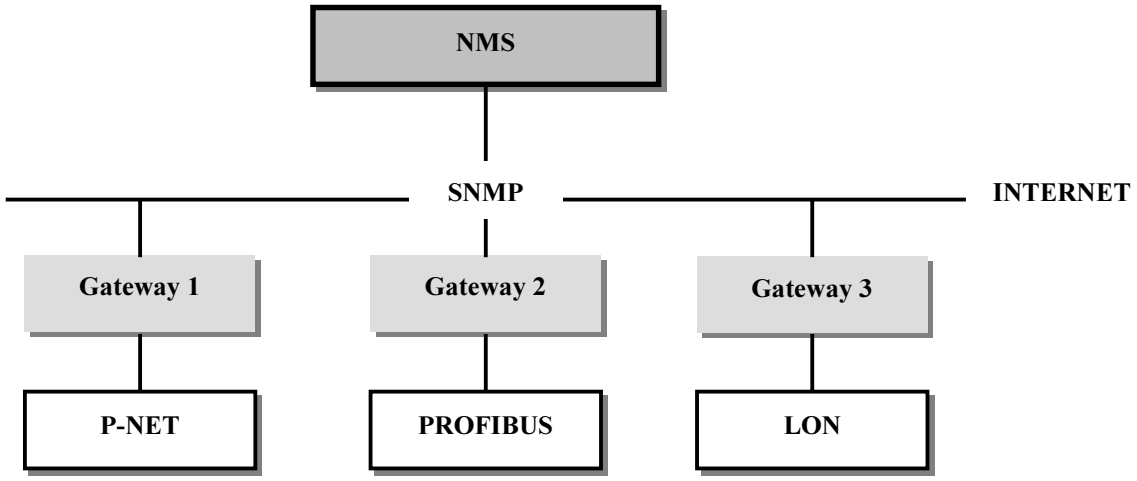


Abb. 4.6 Ein Netzmanagement-System für verschiedene Feldbusse

Dieses Problem kann vermieden werden, wenn das Gateway in zwei Teile unterteilt wird, nämlich in den Master-Agent (MA) und Sub-Agent (SA). Der Master-Agent ist der Teil, der mittels SNMP mit der Außenwelt in Verbindung tritt und eine beliebige Anzahl von Sub-Agenten und damit auch verschiedene Feldbusse verwaltet. Er soll unabhängig vom verwendeten Feldbussystemen aufgebaut werden. Da er anhand der Anforderungen der jeweiligen NMS die Entscheidung trifft, wann und wie ein Wert eines Objekts gelesen oder aktualisiert wird, besitzt er den größten Teil der Gateway-Intelligenz.

Der Sub-Agent ermöglicht den Zugriff auf den Feldbus. Für jedes Feldbussystem muß somit ein Sub-Agent existieren und jeder Sub-Agent ist nur für einen Feldbus zuständig. Der Sub-Agent soll außerdem nur mit dem Master-Agent kommunizieren und dadurch von der Außenwelt und von SNMP unabhängig sein. Damit kann man für die Kommunikation zwischen dem Master-Agent und Sub-Agent statt SNMP ein einfacheres, insbesondere schnelleres Protokoll verwenden.

Durch die Verteilung der Aufgabe des Agenten ist das System modularer und somit auch flexibler geworden. Wenn man neue Feldbussysteme verwalten will, braucht man nicht mehr den ganzen Agent auszuarbeiten, sondern nur den feldbusabhängigen Sub-Agent.

Da die Kommunikation zwischen dem Master-Agent und allen Sub-Agenten über einem einheitlichen Protokoll abläuft und für den Master-Agent die feldbusspezifische Aktivitäten verborgen bleiben, kann man jederzeit ein Sub-Agent für ein neues Feldbussystem an einem Master-Agent anhängen. Außerdem kann man durch diesen modularen Aufbau Änderungen an einer Seite ohne Auswirkung auf die andere Seite vornehmen.

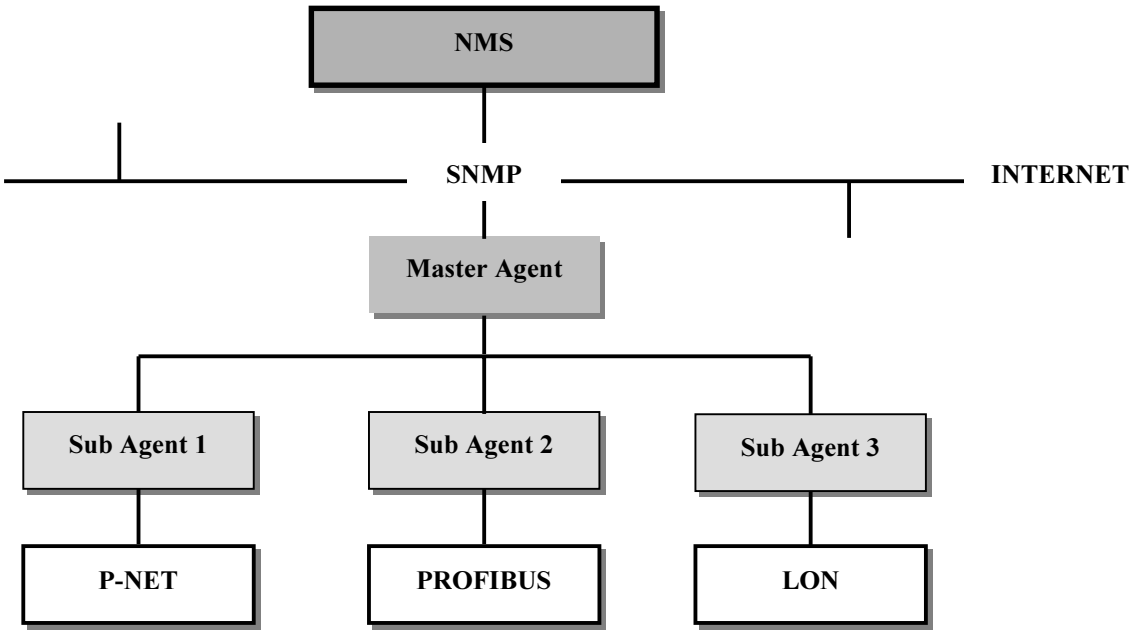


Abb. 4.7 Netzmanagement mit einem geteilten Agent

Da die Kommunikation zwischen NMS und dem Master-Agent auf SNMP-Protokoll basiert, kann der Master-Agent mit jeder SNMP-fähigen NMS kommunizieren. Als NMS wird ein kommerzielles Tool verwendet.

In weiteren Kapiteln werden Master-Agent und die Kommunikation zwischen dem Master- und Sub-Agent erklärt. Da der Sub-Agent für P-NET die Hauptaufgabe dieser Diplomarbeit ist, wird er in den nächsten Kapiteln sehr ausführlich beschrieben.

4.4 Master-Agent

Der Master-Agent dient für die Weiterleitung der Anforderungen der NMS an den entsprechenden Sub-Agent. Wenn beim Master-Agent eine Anfrage der NMS über verwaltetes Objekt einlangt, sucht der Master-Agent den entsprechenden Sub-Agent, welcher das Feldbussystem des gesuchten Objekts verwaltet und falls nötig die Anfrage in entsprechender Form weiterschickt. Nachdem der Sub-Agent die gesuchten Daten entweder direkt aus dem Feldbus oder aus einem eigenen Puffer geholt hat, schickt er sie zum Master-

Agent zurück, der die Daten wieder über SNMP an die NMS zurückschickt. Die Details über diesen Kommunikationsablauf wird in dem Kapitel 4.7 behandelt.

Wie bei Netzmanagment-Agents üblich ist, besitzt der Master-Agent auch eine hierarchisch aufgebaute Datenstruktur, namens Management Information Base (MIB). Diese MIB besteht aus drei Unterbäumen. In einem dieser Unterbäume sind alle nötigen Informationen der verwalteten Objekten gespeichert. Nachdem eine Verbindung zwischen dem Master-Agent und einem Sub-Agent hergestellt wurde, fragt der Master-Agent über einen Konfigurator den Sub-Agenten zuerst nach der Anzahl der verwalteten Objekte. Danach holt der Konfigurator die Daten über die Eigenschaften jedes Objekts und trägt sie in die entprechenden Zeilen der MIB ein. Der Ablauf dieser Initialisierungphase wird im Kapitel 4.7 genauer dargestellt.

Der Konfigurator stellt die Verbindung zu einem bestimmten, vom Benutzer ausgesuchten Sub-Agent her. Mit Hilfe dieses Konfigurators können jene der vom Sub-Agent angebotenen Objekten von dem Benutzer ausgewählt werden, die für das Netzmanagement interessant sind. Der Konfigurator ermöglicht auch händische Bearbeitung der vom Sub-Agenten erhaltenen Daten. Die Objektdaten werden dann vom Konfigurator in eine "MIB"-Datei abgespeichert, aus der der Master-Agent den entsprechenden Teilbaum in der MIB aufbaut.

Diese MIB bietet eine weitere Möglichkeit für die NMS, um die Werte der Objekte abzufragen. Die Werte der verwalteten Objekte werden in bestimmten Intervallen aus dem Feldbus gelesen und in die MIB geschrieben. Wenn dieser Wert den Anforderungen der NMS entsprechend aktuell genug ist, kann er von der NMS verwendet werden und somit bleibt eine Abfrage aus dem Feldbus erspart.

In der MIB des Master-Agenten gibt es auch einen Unterbaum mit einer Tabelle, die die folgenden Einträge hat:

- IP-Adresse eines Sub-Agents
- Portnummer eines Sub-Agents

Durch diese Tabelle weiß der Master-Agent, wo sich die Sub-Agents befinden. Die Einträge in dieser Tabelle können aus einer Konfigurationsdatei geholt werden. Außerdem hat der Master-Agent einen bestimmten Port, an dem ein Sub-Agent sagen kann, daß er an einer Adresse auf die Verbindung mit dem Master-Agent wartet. Die IP-Adresse und Portnummer des jeweiligen Sub-Agenten wird, falls sie noch nicht existieren, auch in die Liste eingetragen. Dieser Vorgang wird als SAR (Sub-Agent Registration) bezeichnet. Eine Anmeldung auf diesem Port bedeutet nur, daß der Sub-Agent existiert, und nicht, daß er automatisch beim Master-Agent angeschlossen wird. Der Master entscheidet selber, wann eine Verbindung zustande kommen soll.

Der letzte Unterbaum beinhaltet eine Liste aller möglichen Arten der Objekte, wie zum Beispiel Länge, Feuchtigkeit, Gewicht, Spannung, Strom usw.

4.5 Sub-Agent

Der Sub-Agent ist für den Feldbus-Zugriff zuständig. Deshalb muß für jedes Feldbussystem ein eigener Sub-Agent implementiert werden. Die Sub-Agents können in passive und aktive Sub-Agents unterteilt werden. Die passiven Sub-Agents haben nur die Aufgabe, gemäß den vom Master-Agent kommenden Anforderungen auf den Feldbus zu zugreifen und den Wert des gefragten Objekts entweder zu lesen oder zu überschreiben. Die aktiven Sub-Agenten können zusätzlich über den Master-Agent auf andere Sub-Agenten und somit auf andere Feldbussystemen zugreifen. Da sich diese Diplomarbeit nur mit einem passiven Sub-Agent beschäftigt, werden im Folgenden nur die passiven Sub-Agents erläutert. Sie werden nachfolgend auch nur Sub-Agent genannt.

Zuerst soll mittels eines Konfigurators die jenigen Objekte aus dem Feldbus ausgewählt werden, die für die Netzmanagement erforderlich sind (siehe Abb. 5.8). Da können die Objekte entweder manuell oder durch eine Autodetect-Funktion selektiert werden.

Bei diesem Auswahl-Verfahren, können außer feldbusspezifischen Daten, die für den Zugriff auf das ausgewählte Objekt im Feldbussystem erforderlich sind, auch folgende Angaben über dem Objekt gemacht werden:

- Typ des Wertes des Objekts
- Beschreibung des Objekts
- Art des Objekts
- Ort des Objekts
- ...

Da aber diese Angaben meistens nur durch Benutzereingriffe zugänglich sind, müssen sie nach einem Autodetect-Auswahlvorgang manuell hinzugefügt werden.

Alle Angaben werden dann in einer ASCII-Textdatei gespeichert. Der Grund für die Speicherung in eine ASCII-Textdatei statt in eine Datenbank ist, daß der Zugriff auf eine Textdatei viel schneller erfolgen kann, die Textdateien weniger Speicherplatz brauchen, für das Lesen und Schreiben auf Texdateien weniger Programmieraufwand erforderlich ist und daß sie mit einem einfachen Text-Editor bearbeitbar sind.

Bei der Initialisierung des Sub-Agenten wird diese Datei gelesen und jedes Objekt wird in den Sub-Agenten aufgenommen. Dabei sollte der Sub-Agent über einen Parameter auf eine bestimmte Konfigurationsdatei parametierbar und nicht auf eine fixe Datei eingestellt sein.

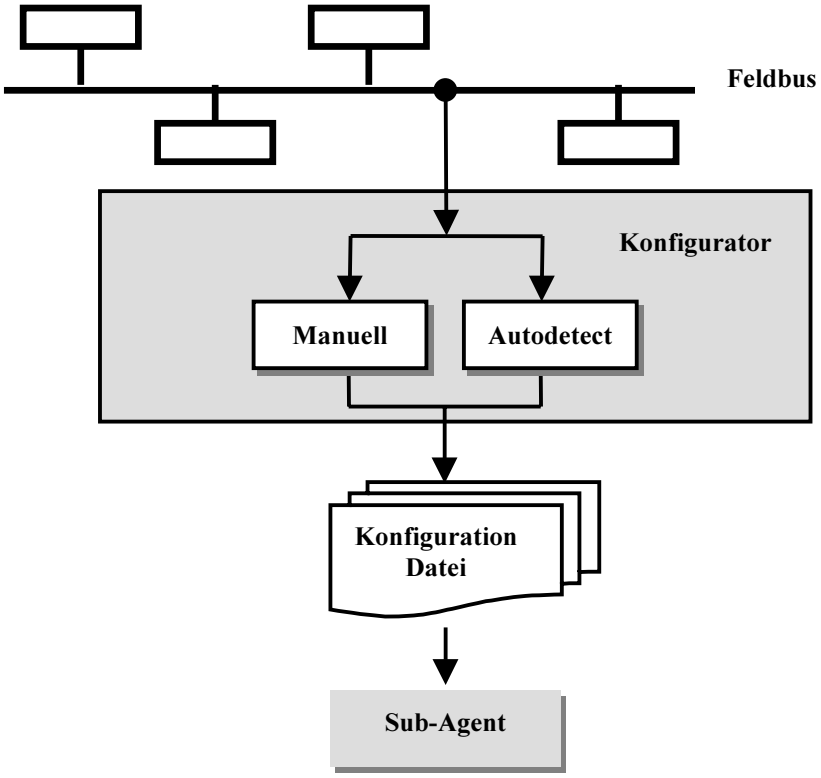


Abb. 4.8 Initialisierung des Sub-Agents

Der Sub-Agent besitzt keine Datenbank, wie der Master-Agent. Er speichert alle Objekte dynamisch in eine Liste und numeriert sie durchgehend mit einer ID-Nummer. Der Master-Agent spricht diese Objekte nur über ihre ID-Nummer an. Wenn mehrere Zeilen des Sub-Agents auf dasselbe Netzwerk-Objekt des Feldbusses verweisen, soll man zwei Listen definieren, um Inkonsistenzen und einen hohen Speicherbedarf zu verhindern. Auf der ersten Liste stehen in jeder Zeile die ID-Nummer eines Objekts und ein Zeiger, der auf das eigentliche Objekt auf der anderen Liste hinweist. Die Abbildung 4.9 zeigt eine Darstellung der Speicherung der Objekte im Sub-Agent.

Bei der Konfiguration des Master-Agents sollte der Sub-Agent dem Master-Agent-Konfigurator mitteilen können, welche Objekte er anzubieten hat. Über dem Konfigurator des Master-Agents wird wieder eine Auswahl getroffen, so daß nur wirklich für das Netzmanagement relevante Objekte dem Manager zur Verfügung gestellt und von ihm verwaltet werden. Die Abbildung 4.10 zeigt dieses schrittweise Selektionsverfahren. Der Master-Agent kennt nicht die physikalische Adresse des verwalteten Objekts, sondern nur seine ID-Nummer. Nur über dieser ID-Nummer kann ein Objekt vom Master-Agent angesprochen werden. Um eine aufeinanderfolgende Abfrage aller Objekte zu ermöglichen, müssen die ID-Nummern durchgehend vorhanden sein und es darf keine Lücken in dieser Reihenfolge geben.

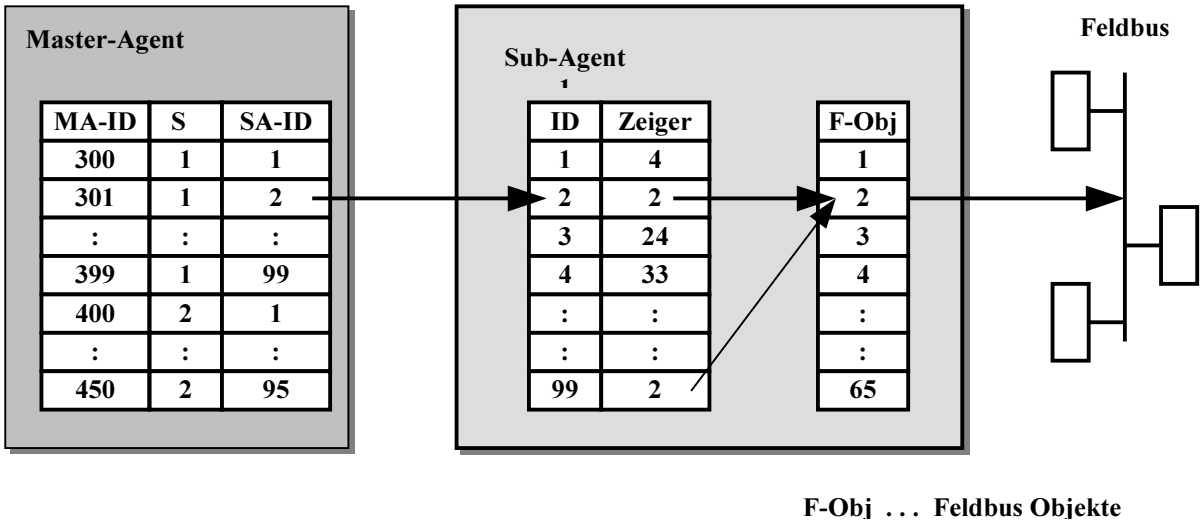


Abb. 4.9 Mögliche Speicherung der Netzwerk-Objekte im Sub-Agent

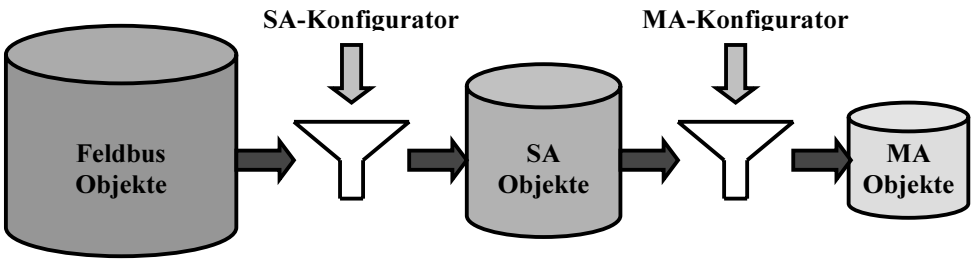


Abb. 4.10 Schrittweise Selektion der verwalteten Objekte [KKM97]

Der Sub-Agent soll mehrere Master-Agenten bedienen können. Er meldet sich nach seiner Initialisierung bei bekannten Master-Agenten an, deren IP-Adresse und Portnummer von einer Initialierungsdatei gelesen werden, wobei der Sub-Agent nur dem Master-Agent mitteilt, wo er zu finden ist. Dieser Vorgang wird SAR (Sub-Agent Registration) genannt. Danach wartet er auf eine Verbindungsanforderung eines Master-Agent. Nachdem eine Verbindung aufgebaut und der Master-Agent initialisiert wurde, wartet der Sub-Agent auf weitere Anforderungen des Master-Agents. Wenn eine Anforderung kommt, führt er dann den erforderlichen Feldbus-Zugriff durch und liest oder setzt den Wert des gefragten Objekts. Beim Lesen kann der Sub-Agent eventuell auch den in einem Puffer gespeicherten Wert zurückgeben. Dafür muß der gespeicherte Wert gemäß der Anforderung des Master-Agenten aktuell genug sein. Die ganze Kommunikation zwischen Master-Agent und Sub-Agent wird im Kapitel 4.7 genauer beschrieben.

4.6 Verbindung zwischen dem Master- und Sub-Agent

Der Master- und alle Sub-Agenten können entweder auf dem selben Rechner laufen oder auf verschiedenen Orten plaziert werden. Wenn sie sich alle auf dem selben Rechner befinden, kann die Kommunikation zwischen dem Master- und den Sub-Agenten sehr schnell und mit einfachen Mechanismen, wie Shared Memory, DLLs oder OLE-Automation, erfolgen. Diese Lösung bringt aber auch mehrere Nachteile mit sich. Allein die Abhängigkeit von einem einzigen Rechner macht das System sehr unzuverlässig und inflexibel. Wenn dieser Rechner außer Betrieb ist, fällt das ganze System aus. Außerdem muß man für jeden Feldbus entsprechende Einsteckkarten und die benötigten Treiber installieren, was bei einem herkömmlichen Rechner nach einer gewissen Grenze diverse Probleme verursachen kann. Wenn man übrigens mehrere Feldbussysteme in verschiedenen Orten verwalten will, müssen diese Systeme mit diesem Rechner verbunden sein, was für große Entfernungen wirtschaftlich nicht vertretbar sein wird.

Deswegen sollen Master- und Sub-Agenten auf unterschiedlichen Rechnern laufen können. Das erfordert einen Mechanismus für die Kommunikation zwischen den Rechnern. Da die Kommunikation möglichst schnell und ohne großen Aufwand erfolgen soll, scheiden mächtige aber auch zu komplexe Mechanismen wie COBRA aus. Am geeignetsten haben sich Sockets erwiesen, weil diese sehr einfach zu programmieren und von jedem Betriebssystem unterstützt sind. Außerdem brauchen Sockets relativ geringere System-Ressourcen als andere Mechanismen.

Ein Socket ist ein Punkt-zu-Punkt Kommunikationsendpunkt mit einem Namen und einer Netzwerkadresse. Berkeleys Sockets sind das erste Protokoll für die Kommunikation über TCP/IP. Sie wurden 1981 als Unix BSD 4.2 Interface, für eine Unix-zu-Unix Interprocess Communication (IPC), vorgestellt. Heute werden Sockets von jedem Betriebssystem unterstützt. Die Windows Socket API, kurz WinSock, ist eine Multivendor Spezifikation, welche die Benützung von TCP/IP unter Windows standardisiert. In Unix sind Sockets Teil des Kernel, während bei MS-DOS oder Windows, Sockets in Form von Bibliotheken bereitgestellt werden.

Ein Socket besteht aus zwei Teilen:

1. Einer IP-Adresse

2. Einer Port Nummer.

Eine Internet Adresse (IP-Adresse) ist eine 32 Bit Zahl, die eindeutig einen Internet-Host identifiziert. Sie kann entweder in Form eines "Dotted String", oder als Domain Name vorliegen. Die Internet Adresse eines Rechners ist zum Beispiel in der Form "128.130.80.191" ebenso gültig, wie "pc191.ict.tuwien.ac.at".

Ein Port ist ein Verbindungspunkt zu einer Anwendung auf einem Host Rechner und wird durch eine 16-bit Nummer repräsentiert. Man beachte, daß die Portnummern 0 bis 1023 für festgelegte Dienste (Email, FTP, HTTP) reserviert sind.

Es gibt drei Arten von Sockets: Datagram, Raw und Stream Sockets. Datagram Sockets finden bei Gebrauch des User Datagram Protocol (UDP) Verwendung, Raw Sockets hingegen bei der Benützung des Internet Control Message Protocols (ICMP). Die größte Verbreitung haben die sogenannten Stream Sockets, die für das Transfer Control Protocol (TCP) und somit für die verbindungsorientierte Kommunikation verwendet werden. Da die Kommunikation zwischen dem Master- und Sub-Agent auf dem zuverlässigeren TCP basieren soll, verwenden wir die Stream Sockets.

Leider bieten die Sockets nicht ausreichend Datensicherheit. Deswegen braucht man für eine sicherheitskritische Datentransfer zusätzliche Sicherheitsmaßnahmen. Da aber solche Maßnahmen nicht zu dieser Diplomarbeit gehören, soll nur eine zum Teil leere Zwischenschicht eingezogen werden, in der in weiterer Folge eine Krypto-Schicht eingezogen werden kann.

Es sind zwei Socket-Verbindungen zwischen einem Master-Agent und Sub-Agent erforderlich: eine für die Sub-Agent Registration (SAR) und eine für den Datenaustausch zum Zweck des Netzmanagements.

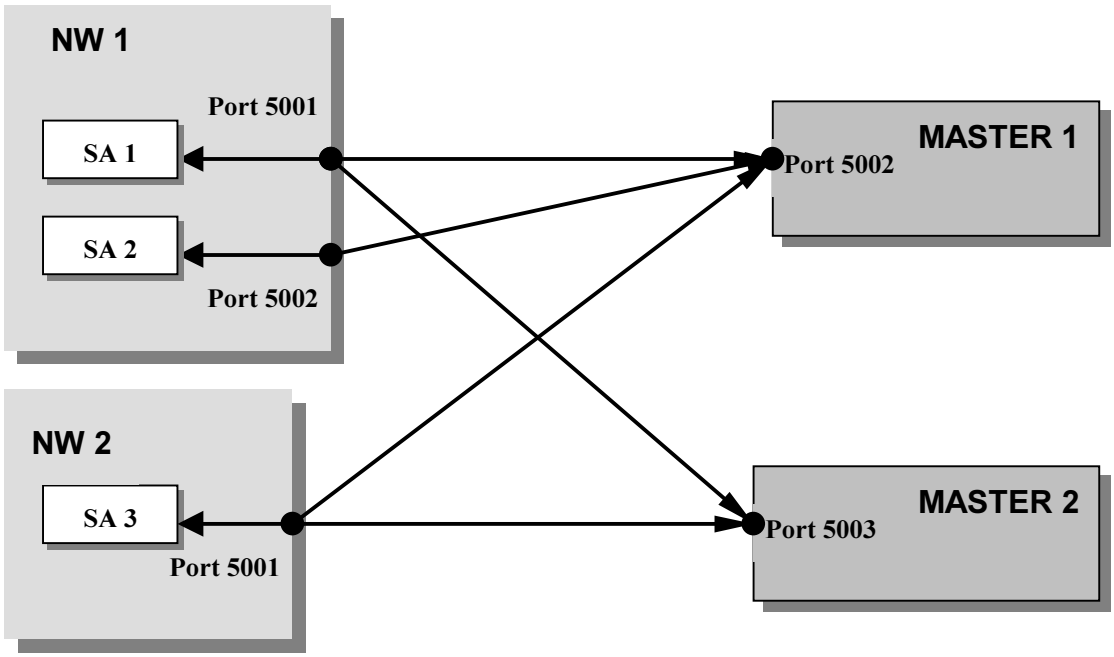


Abb. 4.11 Beispiel für Netzmanagmentsystem

Für SAR hört der Master-Agent auf einen bestimmten Port und wartet auf eine Sub-Agent-Anmeldung. SAR ist nur möglich, wenn dem Sub-Agent diese Portnummer mit der IP-Adresse des Master-Agents bekannt ist. Falls der Sub-Agent die Adresse des Master-Agenten nicht kennt und sich beim Master-Agent nicht anmelden kann, hört der Sub-Agent auf einen anderen Port und wartet auf eine Verbindungsanforderung vom Master-Agent. Da muß der Master-Agent natürlich irgendwie die Adresse des Sub-Agenten kennen, damit eine Verbindung zustande kommen kann. Da mehrere Sub-Agenten pro Netzwerkknoten laufen

können, während es aber nur ein Master-Agent pro Knoten geben kann, muß der Port über den der Sub-Agent vom Master-Agent angesprochen wird, frei konfigurierbar sein (siehe Abb. 4.11).

4.7 Kommunikationsprotokoll zwischen dem Master- und Sub-Agent

Bei diesem Gateway sind 3 Schnittstellen zu unterscheiden. Die zwei nach außen hin sind durch die RFC´s über SNMP und die Beschreibung des Feldbussystems genau beschrieben. Über die interne Schnittstelle zwischen dem Master-Agent und Sub-Agent werden die an den Feldbus gehenden Anfragen in der von der Ausprägung des selben unabhängigen Form übermittelt.

Es gibt einige Standards für das Design verteilter Agenten, wie zum Beispiel:

- SNMP Multiplexing (SMUX) Protokoll [RFC1227],
- SNMP Distributed Program Interface (SNMP-DPI) [RFC1228] oder DPI 2.0,
- Agent Extensibility (AgentX) Protokoll [RFC2257].

Leider sind diese Standards für unsere Implementierung zu komplex und haben auch andere Anforderungen für den Aufbau verteilter Agenten. Das SMUX-Protokoll, das gleich ein Jahr nach dem SNMPv1-Protokoll erschienen ist, verwendet das SNMP-Protokoll für die Kommunikation zwischen den verteilten Agenten, was aber zu einem Datenoverhead und zu einer unnötigen Leistungsverminderung bei den protokollabhängigen Tasks führt. SMUX kann nur für Sub-Agenten implementiert werden, die sich auf dem selben Rechner befinden. Außerdem erlaubt SMUX nicht, daß eine Tabelle in einer MIB von mehreren Agenten gleichzeitig verwendet werden kann.

DPI verwendet ein spezielles Protokoll für die Kommunikation zwischen dem Master-Agenten und Sub-Agenten, welches auf TCP basiert. Bei DPI ist es leider unmöglich, manche Zeilen der MIB bestimmten Sub-Agenten zuzuordnen. Die Sub-Agenten sollen die ganze MIB-Struktur implementieren.

Im Gegensatz zu AgentX hat unser System eine fixe MIB im Master definiert. Die Sub-Agenten liefern nur die Daten, die der Master in einer vorgegebenen Struktur ablegt. Das AgentX-Protokoll ermöglicht auch beliebige, eventuell überschneidende Unterbäume verschiedener Sub-Agenten. Deshalb ist das AgentX-Protokoll das Geeignetste von diesen drei Protokollen. Dieses Protokoll war aber noch nicht standardisiert, als wir angefangen haben, unser System zu realisieren. Deswegen konnten wir es nicht anwenden.

Um Programmieraufwand und Daten-Overhead bei der Kommunikation möglichst gering zu halten, wurde ein zu unserer Aufgabe zugeschnittenes Protokoll hergestellt. Die Kodierung dieses Protokolls soll aber auf einen Standard basieren, damit die zu verschiedenen Zeiten entwickelten Module problemlos miteinander kommunizieren können. Für diesen Zweck

eignet sich die OSI-Sprache Abstract Syntax Notation One (ASN.1), die im Kapitel 2.5 genauer erklärt wurde, weil sie ein bekannter und erprobter Standard ist und auch für die Kodierung von SNMP-Daten benutzt wird.

Die Kommunikationsablauf für die Schnittstelle zwischen Master- und Sub-Agenten besteht hauptsächlich aus zwei Teilen:

- Sub-Agent Registration (SAR),
- Datenaustausch zwecks Netzmanagement.

Der erste Teil wurde vorher beschrieben. Der zweiten Teil erfüllt die Hauptaufgabe des Netzmanagements. In diesem Teil werden die vom Master-Agenten an den Feldbus gehenden Anfragen und vom Sub-Agenten zurückkommenden Antworten in einer standardisierten Form, nämlich in ASN.1, kodiert und zwischen Agenten gesendet.

Für das Netzmanagement brauchen wir einige Operationen, die zum Austausch von Verwaltungsinformationen zwischen dem Master- und Sub-Agenten dienen. Es gibt fünf Anfragen, die von einem Master-Agenten zu einem Sub-Agenten geschickt werden:

1. `ConnectRequest:` Diese Anforderung besagt, daß der Master-Agent eine Kommunikation mit dem Sub-Agent aufbauen will. Dabei wird auch die Anzahl der von diesem Master-Agent verwalteten Sub-Agents mitgeschickt.

2. `GetMaxNumberRequest`: Mit dieser Anforderung fragt der Master-Agent den Sub-Agenten, wieviele Objekte er insgesamt zu verwalten hat. Diese Anfrage dient zur Initialisierung des Master-Agenten.

3. `GetRequest`: Mit dieser Anforderung kann ein Master-Agent den Wert eines verwalteten Objekts aus dem Sub-Agenten erhalten. In dieser Operation werden die ID-Nummern des gefragten Objekts, mit der das Objekt identifiziert wird, und eine gewisse Delta-Time mitgeschickt. Diese Delta-Time besagt, wie alt ein in dem Puffer des Sub-Agenten gespeicherten Wert höchstens sein kann, damit dieser zum Master-Agenten gesendet werden kann, ohne auf den Feldbus zuzugreifen. Somit kann ein unnötiger Datenverkehr auf dem Feldbus vermieden werden. Falls der gespeicherte Wert nicht aktuell genug ist, soll dieser aus dem Feldbus geholt werden. Die Abbildung 4.12 zeigt die Protokollsequenz für das Lesen eines Wertes. Wie groß diese Delta-Time sein soll, damit eine optimale Aktualisierung der Daten erfolgen kann, wird vom Master-Agent durch bestimmte Algorithmen ermittelt.

4. `GetDetailRequest`: Mit dieser Anfrage fragt der Master-Agent nach den Eigenschaften des verwalteten Objekts, wie die Beschreibung und Status des Objekts oder der ASN.1 Typ seines Wertes sind. Diese Daten werden dann in die MIB eingetragen.

5. `SetRequest:` Mit Hilfe dieser Anfrage kann der Master-Agent gemäß den Anforderungen der NMS den Wert eines Objekts verändern. Da werden die ID-Nummer und der Wert des Objektes gesendet. Da soll der neue Wert, der als ein String geschickt wird, dem eigentlichen Typ des Wertes entsprechen.

Für jede Anforderung vom Master-Agent gibt es auch eine entsprechende Antwort-Meldung (Response) vom Sub-Agent, die auch einen Fehlerwert mitschickt. Dieser Wert ist Null, wenn kein Fehler aufgetreten ist, oder enthält einen feldbusabhängigen Fehlercode.

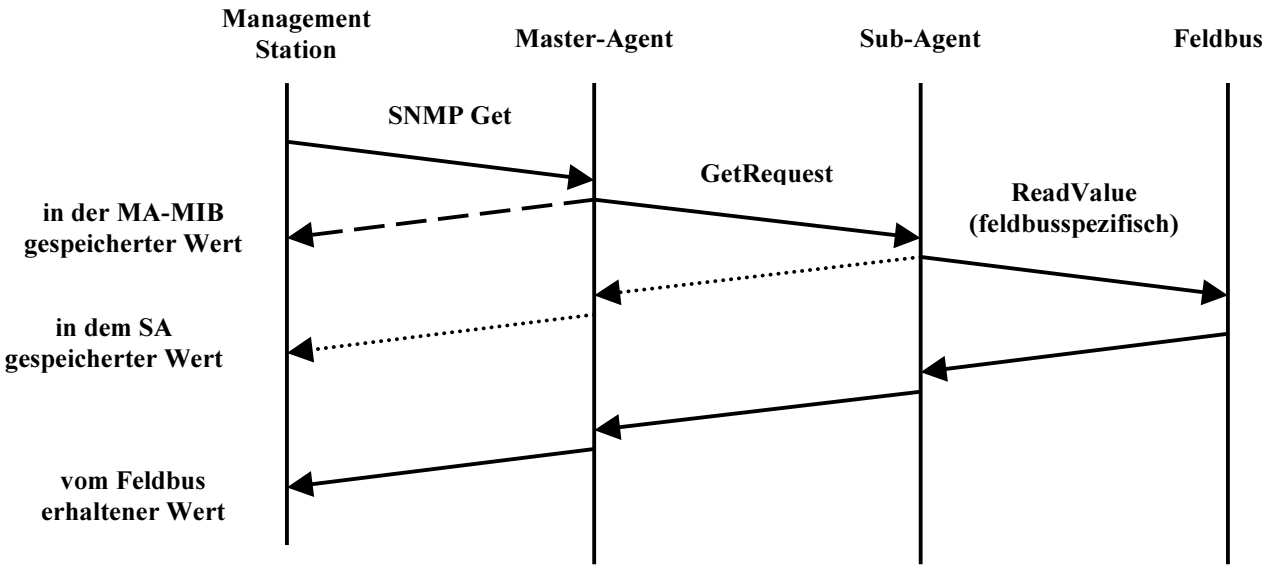


Abb. 4.12 Protokollsequenz für das Lesen eines Wertes [KKM97]

Die Antwort `GetResponse` enthält außerdem einen Status-Teil, welcher einer von folgenden ganzzahligen Werten haben kann:

- 0 : Dieser Wert bedeutet, daß der gelieferte Wert direkt aus dem Feldbus stammt und dabei kein Fehler aufgetreten ist. So ist dieser Wert aktuell und höchstwahrscheinlich richtig.

- 1: Wenn der Wert des Status-Teils 1 ist, dann kommt der Wert aus dem Puffer des Sub-Agents. Da hat kein Feldbus-Zugriff stattgefunden.

- 2: Wenn die Messung fehlerhaft war, aber der Wert durch irgendwelche Algorithmen korrigiert werden konnte, dann ist der Status-Wert 2.

- 3: Der Status-Wert 3 bedeutet, daß ein Meßfehler aufgetreten ist und der mitgelieferte Wert wieder ein vorher gespeicherter Wert ist. Die Fehlercode besagt dabei, um welchen Fehler es sich handelt.

- 5: Dieser Status-Wert bedeutet, daß der Wert aus dem Feldbus mit Hilfe der redundanten Sensoren gemessen wurde und deswegen mit Sicherheit stimmen sollte.

- 6: Wenn eine Antwort den Status-Wert 6 erhält, dann ist bei dem Feldbus-Zugriff ein Fehler aufgetreten und es gibt auch kein gespeicherter Wert. Deswegen ist der mitgelieferte Wert mit Sicherheit ungültig.

Es existiert außerdem noch eine weitere Meldung, nämlich `Abort-Indication`, die ein Kommunikationspartner zu der anderen Seite der Verbindung sendet, wenn die Verbindung abgebrochen werden wird. Diese Meldung enthält einen Integer-Wert, der auf den Grund des Kommunikationsabbruches hinweist.

Wenn der Master-Agent sich beim Sub-Agent anmelden will, dann baut der MA-Konfigurator zuerst eine Socketverbindung mit demjenigen Port auf, auf dem der Sub-Agent auf die Verbindung mit einem Master-Agent wartet. Danach sendet er eine `ConnectRequest` zum Sub-Agent mit der Anzahl aller von ihm verwalteten Sub-Agents. Falls er eine positive Antwort, nämlich `ConnectResponse` mit einem Errorcode 0, vom Sub-Agent bekommt, schickt er als Nächstes die Anfrage `GetMaxNumberRequest,` um die Gesamtanzahl der verwalteten Objekte zu erfahren. Nacher fragt er mit Hilfe der Anfrage `GetDetailRequest` nach den Eigenschaften der verwalteten Objekten. Nachdem alle Objekte durchgefragt wurden, wird die Verbindung mit einer `AbortIndication`-Meldung vom MA-Konfigurator getrennt. Nach einer Selektion der Objekte, werden die Eigenschaften der Objekte in eine Konfigurationsdatei gespeichert, aus der der Master-Agent diese Eigenschaften in die zugehörigen Teilen der MIB einträgt. Danach baut der Master-Agent eine Verbindung mit dem Sub-Agent auf dem selben Port wie sein Konfigurator auf. Nachdem der Master-Agent vom Sub-Agent die Meldung `Connect-Response` mit dem Errorcode 0 erhalten hat, kann er die Werte der Objekte entweder mit einer `GetRequest` lesen oder mit einer `SetRequest` setzen. Die Abbildung 4.13 zeigt die Initialisierungsphase zwischen dem Master-Agent-Konfigurator und Sub-Agent bzw. die Abbildung 4.14 stellt ein Beispiel für die Kommunikation zwischen dem Master- und Sub-Agent nach der Initialisierung dar.

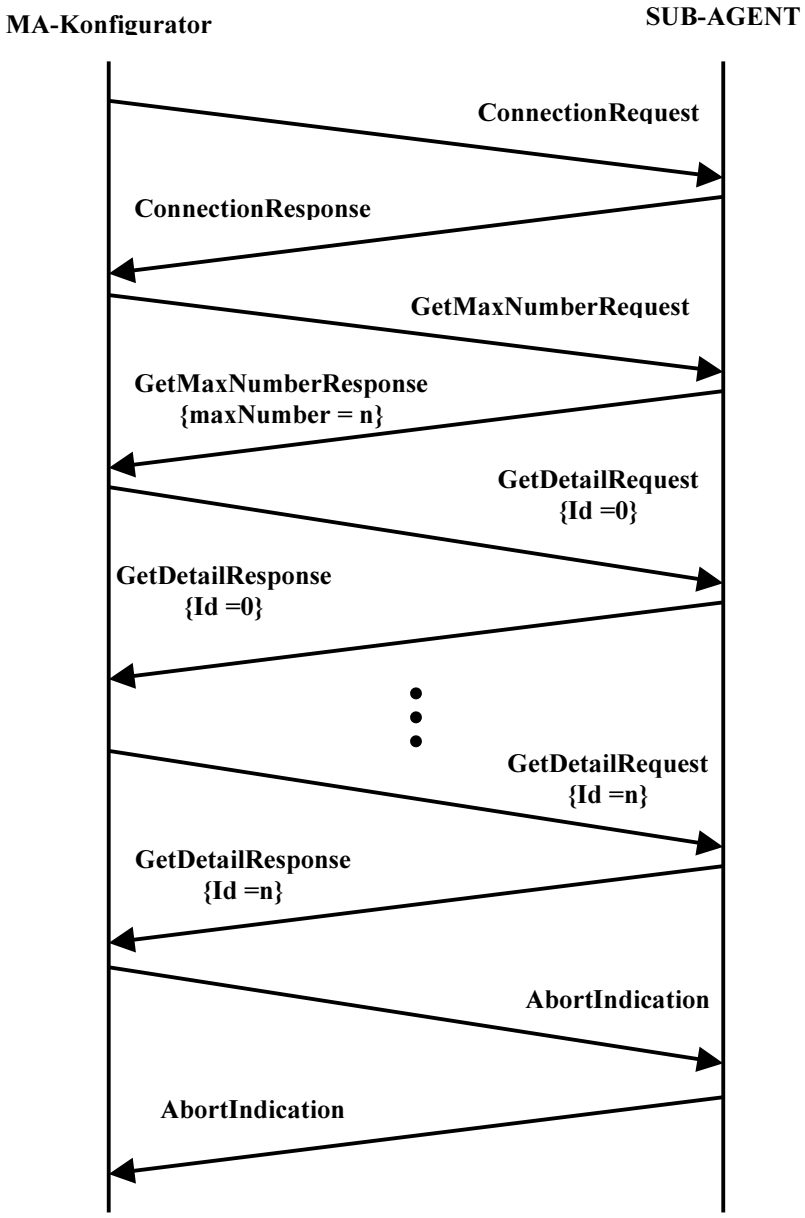


Abb. 4.13 Initialisierungsphase zwischen Master- und Sub-Agent

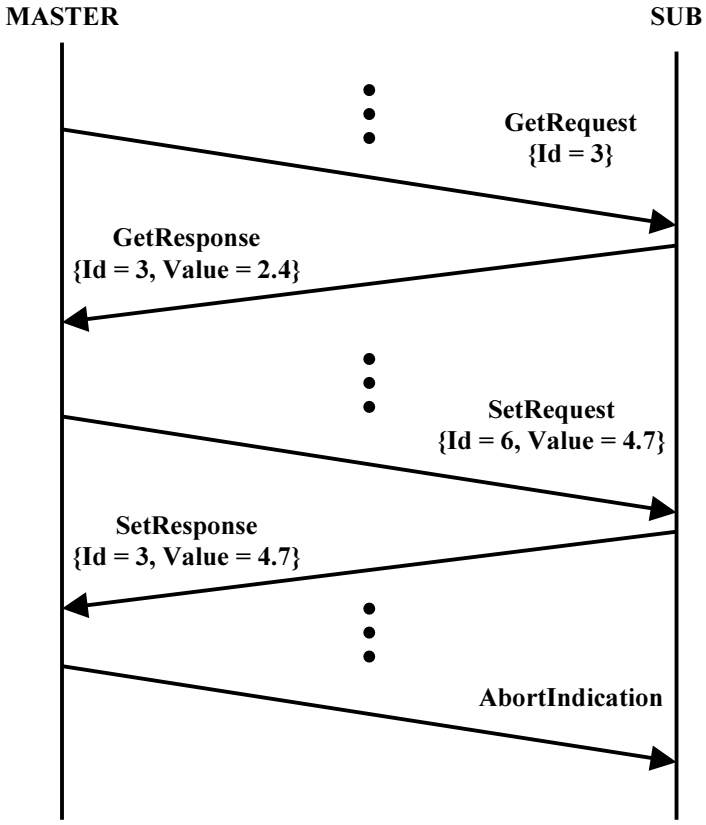


Abb. 4.14 Kommunikation zwischen dem Master- und Sub-Agent nach der Initialisierungsphase

Leider war die Spezifikation des ersten Teils (SAR) noch nicht fertig, als diese Diplomarbeit geschrieben wurde. Der zweite Teil der Kommunikation, kodiert in ASN.1, sieht folgendermaßen aus, wobei die Teile, die nur für einen aktiven Sub-Agent relevant sind, werden kursiv geschrieben:

```
Sub-Master-Kommunikation-SMK DEFINITIONS ::=
Begin

SMK-PDU::= CHOICE
{
      [0] SMK-Connect-Request,
      [1] SMK-Connect-Response,
      [2] SMK-Master-Request,
      [3] SMK-Master-Response,
      [4] SMK-Sub-Request,
      [5] SMK-Sub-Response,
      [6] SMK-Abort-Indication
}

SMK-Connect-Request    ::= SEQUENCE
{
      numberSA                          INTEGER,
      error                             INTEGER
```

```
}

SMK-Connect-Response   ::= CHOICE
{
      noError                          [0]  IMPLICIT NULL,
      noFieldbusConnected              [1]  IMPLICIT NULL,
      anotherMA                        [2]  IMPLICIT NULL,
      generalError                     [32] IMPLICIT NULL
}

SMK-Master-Request     ::= CHOICE
{
      getMaxNumberRequest              [0] IMPLICIT NULL,
      [1] GetRequest,
      [2] GetDetailRequest,
      [3] SetRequest
}

SMK-Master-Response    ::= CHOICE
{
      [0] GetMaxAnzahlResponse,
      [1] GetResponse,
      [2] GetDetailResponse,
      [3] SetResponse
}

SMK-Sub-Request  ::= CHOICE
{
      [0] GetSubValueRequest,
      [1] GetSubMibRequest,
      [2] SetSubValueRequest,
      [3] SetSubMibRequest
}

SMK-Sub-Response ::= CHOICE
{
      [0] GetSubValueResponse,
      [1] GetlSaveMibResponse,
      [2] SetSubValueResponse,
      [3] SetSubMibResponse
}

SMK-Abort-Indication   ::= CHOICE
{
      userAbort                        [0] IMPLICIT NULL,
      systemShutdown                   [1] IMPLICIT NULL,
      otherMatters                     [32] IMPLICIT NULL
}

GetRequest  ::= SEQUENCE
{
      getRequestId                     INTEGER,
      getRequestDeltaTimeMax           GENERALIZED TIME
}

GetDetailRequest ::= SEQUENCE
```

```
{
      getDetailRequestId                INTEGER
}

SetRequest  ::= SEQUENCE
{
      setRequestId                      INTEGER,
      setRequestValue                   OCTET STRING
}

GetMaxNumberResponse ::= SEQUENCE
{
      maxNumber                         INTEGER,
      maxNumberError                    INTEGER
}

GetResponse ::= SEQUENCE
{
      getResponseId                     INTEGER,
      getResponseValue                  OCTET STRING,
      getResponseTime                   GENERALIZED TIME,
      getResponseDeltaTime              GENERALIZED TIME,
      getResponseStatus                 INTEGER,
      getResponseError                  INTEGER
}

GetDetailResponse::= SEQUENCE
{
      getDetailResponseId               INTEGER,
      getDetailResponseKind             INTEGER,
      getDetailResponseDescr            OCTET STRING,
      getDetailResponseType             INTEGER,
      getDetailResponseStatus           INTEGER,
      getDetailResponseError            INTEGER
}

SetResponse ::= SEQUENCE
{
      setResponseId                     INTEGER,
      setResponseError                  INTEGER
}

GetSubValueRequest     ::= SEQUENCE
{
      getSubRequestID                   INTEGER,
      getSubRequestDeltaTime            GENERALIZED TIME
}

GetSubMibRequest ::= SEQUENCE
{
      getMibRequestOID                  OID
}

SetSubValueRequest     ::= SEQUENCE
{
      setSubRequestID                   INTEGER,
```

```
        setSubRequestValue              OCTET STRING
}

SetSubMibRequest ::= SEQUENCE
{
        setSubMibRequestOID             OID,
        setSubRequestValue              OCTET STRING
}

GetSubValueResponse    ::= SEQUENCE
{
        getSubResponseID                INTEGER,
        getSubValueResponseValue        OCTET STRING,
        getSubValueResponseTime         GENERALIZED TIME,
        getSubValueResponseDeltaTime    GENERALIZED TIME,
        getSubValueResponseStatus       INTEGER,
        getSubValueResponseError        INTEGER
}

GetSubMibResponse::= SEQUENCE
{
        getMibResponseOID               OID,
        getSubMibResponseValue          OCTET STRING,
        getSubMibResponseError          INTEGER,
}

SetSubValueResponse    ::= SEQUENCE
{
        setSubResponseID                INTEGER,
        setSubValueResponseError        INTEGER
}

SetSubMibResponse::= SEQUENCE
{
        setSubMibResponseOID            OID,
        setSubMibResponseError          INTEGER
}
END
```

5. KONFIGURATOR VON SUB-AGENT FÜR P-NET

5.1 Allgemeine Eigenschaften des Konfigurators

Der Konfigurator eines Sub-Agenten ist ein Tool, welches zum Auswählen der Objekte des Feldbussystems dient, die während des Netzmanagements von dem Sub-Agent verwaltet werden.

Der Konfigurator hat allgemein folgende Aufgaben:

- automatisches Erkennen und Anzeigen aller verfügbaren Feldbus-Objekte
- automatisches oder manuelles Auswählen der Objekte
- manuelle Eingabe und Modifikation der Eigenschaften der Objekte

Der Konfigurator soll die ausgewählten Objekte und deren Eigenschaften, welche in den folgenden Kapiteln dargestellt werden, in eine Textdatei speichern, damit sie später einfach mit einem Text-Editor verändert werden können.

In Folge dieser Diplomarbeit wurde auch ein Konfigurator für den Sub-Agent für das Feldbussystem P-NET geschrieben, der eine sogenannte OLE-Schnittstelle der Manager Information Base (MIB) von P-NET verwendet, um die Informationen, die für den Zugriff auf ein bestimmtes Feldbus-Objekt nötig sind, aus der MIB erhalten zu können. Die MIB von P-NET wurde bereits im Abschnitt 4.6.2 erläutert.

Der Konfigurator wurde mit der Programmiersprache C++ unter der Verwendung der Entwicklungsumgebung MS Visual C++ 5.0 als eine 32-Bit Applikation für MS Windows 9x/NT geschrieben.

5.2 Initialisierungsphase

Nach dem Aufrufen des Konfigurators wird die Initialisierungsdatei "agentcon.ini" geöffnet, welche sich in dem selben Verzeichnis wie das eigentliche Programm des Konfigurators befinden muß. Falls diese Datei nicht gefunden werden kann, wird der Konfigurator mit einer Fehlermeldung terminiert.

Die Initialisierungsdatei besteht aus zwei Teilen:

1. Types: Hier stehen alle möglichen Typen, die der Wert eines verwalteten Objekts haben kann. Sie werden beim Auswählen eines Objekts in einer Liste dem Benutzer zur Verfügung gestellt.

2. Kinds: In diesem Teil steht die Liste der möglichen Objekttypen, welche besagen, was diese Objekte in Wirklichkeit präsentieren, ob ein Element zum Beispiel einen Schalter, einen Temperaturwert usw. darstellt.

Wenn einer von diesen Teilen nicht existiert, endet der Konfigurator mit einer entsprechenden Fehlermeldung. Die Abbildung 5.1 zeigt ein vereinfachtes Strukturogramm für diese Initialisierungsphase.

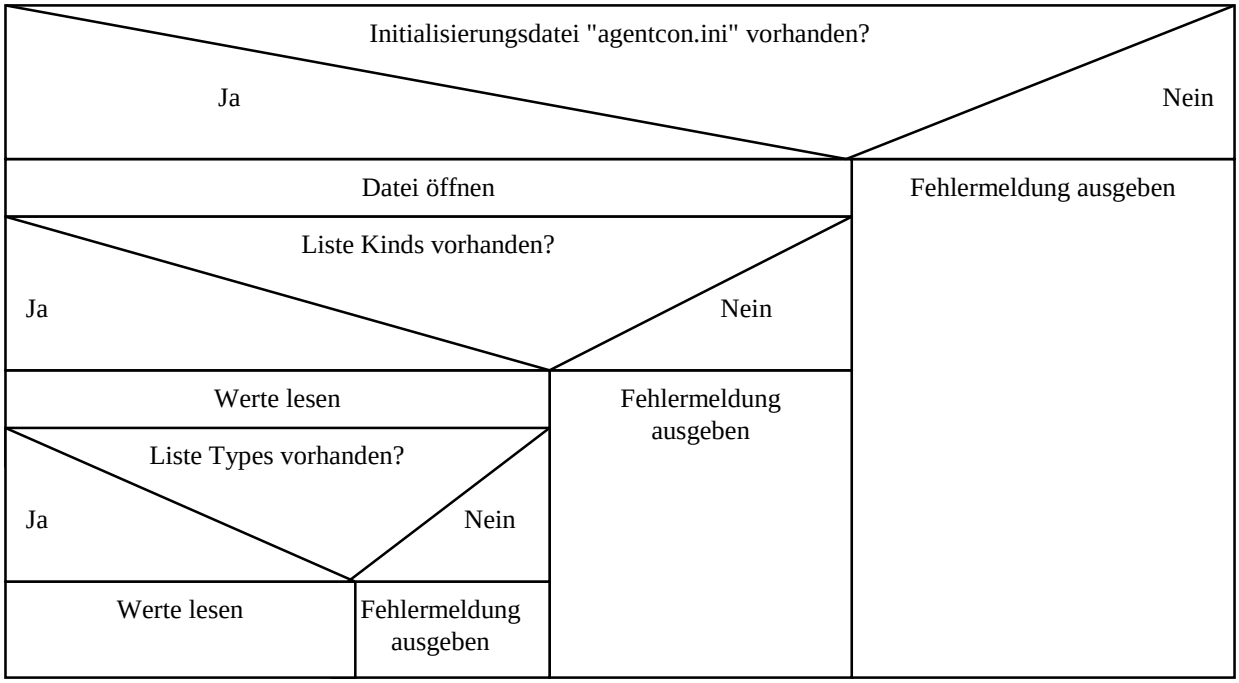


Abb. 5.1 Strukturogramm für die Initialisierungsphase des Konfigurators

5.3 Benutzeroberfläche des Konfigurators

Der Konfigurator besitzt eine graphische Benutzeroberfläche, welche dem Benutzer eine Schnittstelle zum Feldbus zur Verfügung stellt und dadurch das Auswählen der Objekte für das Netzmanagement ermöglicht. Die Abbildung 5.2 zeigt diese Benutzeroberfläche.

Diese Benutzeroberfläche besteht hauptsächlich aus folgenden Teilen:

1. einem Feld, namens "`MIB-View`", der alle Objekte aus der P-NET MIB zeigt

2. einem Feld, genannt "Selected Objects", das alle ausgewählten Objekte in einer baumartigen Struktur zeigt

3. einer Liste, die Logical-IDs aller ausgewählten Objekte, die zur eindeutigen Identifizierung der Objekte dienen.

4. einem Menü-Teil und diversen Schaltflächen.

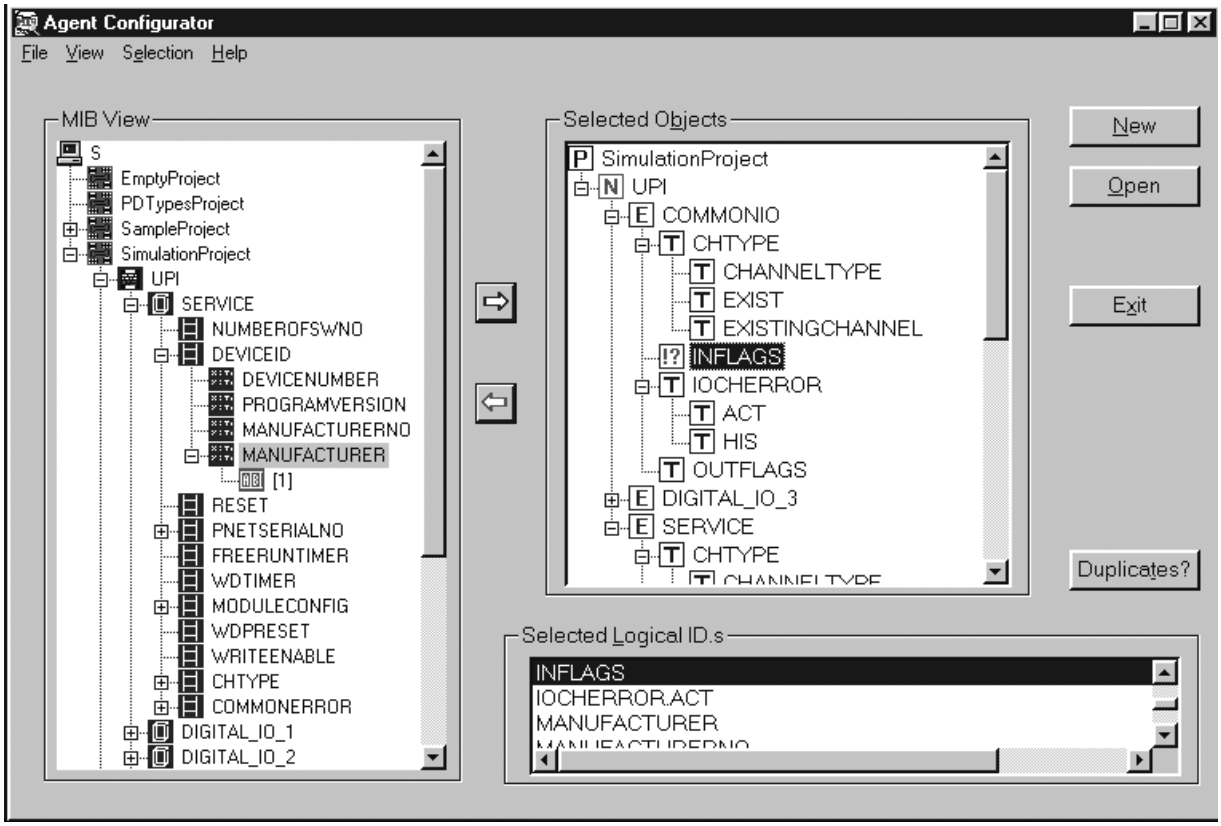


Abb. 5.2 Die Benutzeroberfläche des Konfigurators

Der "MIB-View" Teil des Konfigurators stellt eine graphische Schnittstelle zwischen der P-NET MIB und dem Benutzer dar. Für die Darstellung der MIB-Baumstruktur wurde eine MS Windows OLE Control Extension (OCX), genannt MIBOCX, verwendet, die von der Firma PROCES-DATA A/S mit VIGO mitgeliefert wird.

Die von der Firma Microsoft entwickelten OLE- und OCX-Technologien ermöglichen das Verknüpfen und Einbetten von Objekte oder Komponenten von Windows-Applikationen. Beide verwenden als Basis Component Object Model (COM), welches definiert, wie OLE- und OCX-Objekte miteinander interagieren. Auf ein COM-Objekt kann über eine Schnittstelle von anderen Applikationen zugegriffen werden. Diese Schnittstelle ist eine Sammlung von

verschiedenen Methoden, die den Zugriff auf das entsprechende Objekt ermöglichen. Die OCX-Objekte, die auch ActiveX-Steuerelemente genannt werden, sind COM-Objekte, die eine zusätzliche Schnittstelle besitzen. Diese Schnittstelle funktioniert wie ein Windows-Steuerelement, so daß sie eine sichtbare Benutzeroberfläche zur Verfügung stellt, die in die Benutzeroberflächen anderer Windows-Applikationen eingebettet werden kann. Da aber die OLE-Schnittstelle und MIBOCX nur auf einem PC mit dem Betriebsystem MS Windows 9x/NT lauffähig sind, kann der Konfigurator auch nur auf solchem Computern funktionieren.

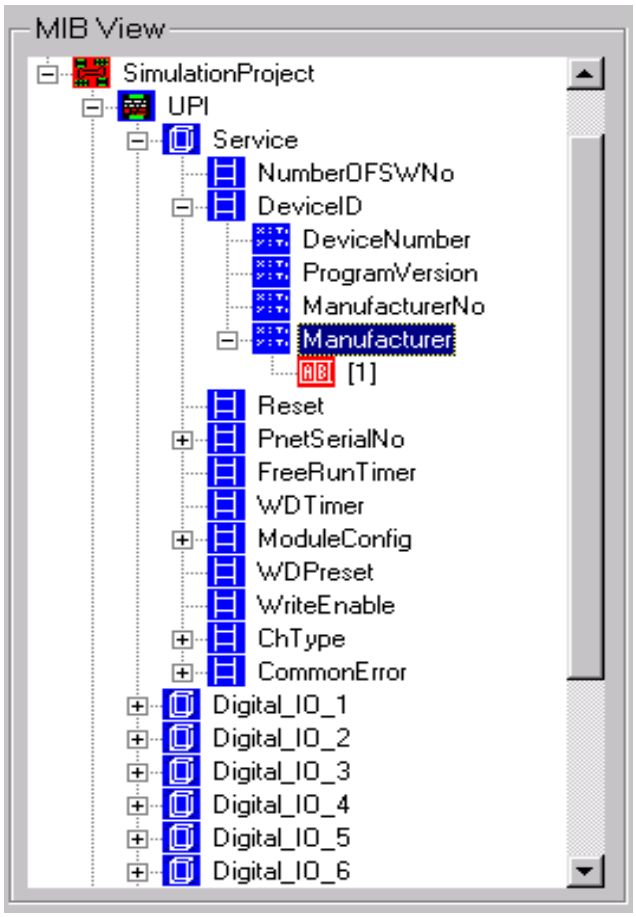


Abb. 5.3 Der "MIB View" Teil des Konfigurators

In dem MIBOCX werden alle Elemente der MIB von P-NET in einer Baumstruktur gezeigt. An dem obersten Teil eines Baums steht immer der Name des jeweiligen Projekts. Unter diesem Projekt befinden sich die Knoten (Nodes), die die verschiedene Channels beinhalten. Unter jedem Channel sind verschiedene Variablen enthalten, die entweder einen einfachen Typ, wie Integer oder String, oder einen zusammengesetzten, wie Array oder Record, besitzen. Die Abbildung 5.3 zeigt eine detaillierte Darstellung dieser Baumstruktur.

Das MIBOCX-Steuerelement ermöglicht auch einen Zugriff auf die MIB Elemente. Durch das Klicken der rechten Maustaste auf ein bestimmtes Element kann man eine Menü mit verschiedenen Funktionen aufrufen, die nur für das ausgewählte Element relevant sind. Durch

solche Funktionen hat man die Möglichkeit, die Eigenschaften der MIB-Objekte zu modifizieren, neue Objekte zu definieren oder bestehende aus der MIB zu löschen.

5.4 Auswählen eines Objekts

Man kann ein Objekt aus dem MIB-View durch folgende Aktionen auswählen:

1. Doppelklicken auf das Objekt,

2. Betätigen des Schaltknopfes mit einem blauen Pfeil,

3. Eingabe der Tastenkombination ALT+S.

Da kann man zwei Arten von Elementen auswählen:

1. Variablen, die einen einfachen Wert zurück liefern. Diese sind solche Variablen mit einfachen Typen oder einzelne Elemente der Variablen mit zusammengesetzten Datentypen.

2. Elemente mit dem Typ Channel.

Falls man ein Element mit dem Channel auswählt, werden alle Variablen, die zu diesem Channel gehören und einen einfachen Wert liefern, automatisch ausgewählt. Dabei soll man in Betracht ziehen, daß bei zusammengesetzten Objekten, wie zum Beispiel bei einem Array, jedes Element dieses Objekts einzeln selektiert wird. Leider sind ein paar neue Einschränkungen für die letzten Versionen von VIGO und MIB-OLE-Schnittstelle von der Seite der Herstellerfirma PROCES-DATA AS implementiert worden, die zum Beispiel eine Abfrage des Wertes eines ganzen Bitarrays ausschließen, obwohl das in früheren Versionen möglich war. Der Benutzer des Konfigurators ist dadurch gezwungen, jedes Bit in einem Bitarray einzeln zu selektieren, was den Aufwand der Arbeit unnötig erhöhen wird. Da aber der Konfigurator in dieser Hinsicht flexibel aufgebaut ist, wird eine Verbesserung dieser Lücke in den zukünftigen Versionen von VIGO und MIB-OLE-Schnittstelle automatisch und ohne Änderung des Quellcodes vom Konfigurator wahrgenommen.

Falls man einen anderen Wert, der keinen einfachen Wert liefert oder nicht den Typ Channel beinhaltet, auswählen will, kommt einfach eine Fehlermeldung.

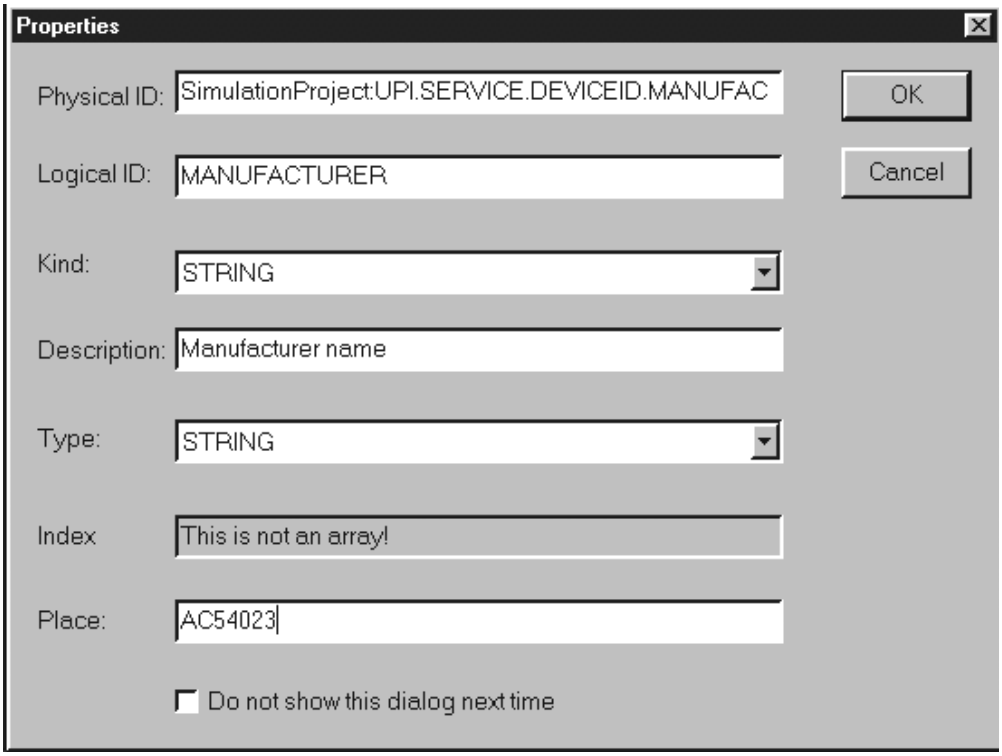


Abb. 5.4 Dialogfenster zur Anzeige der Objekteigenschaften

Beim Auswählen eines Objekts erscheint ein Dialogfenster, namens "Properties", welches in der Abbildung 5.4 gezeigt wurde. Dieses Dialogfenster enthält folgende Felder:

1. `Physical ID`: In diesem Feld erscheint die physikalische Feldbus-Adresse des gewählten Objekts, welche automatisch über die OLE-Interface aus der P-NET MIB erhalten wird. Durch diese Adresse kann über VIGO auf das Objekt zugegriffen werden. Bei einem Array erscheint die Adresse des ersten Elementes dieses Arrays mit der Indexnummer Null. Durch die Änderung der Indexnummer kann ein anderes Element aus diesem Array ausgewählt werden, wobei das Feld `Index` in Betracht gezogen werden muß.

2. `Logical ID`: In dieses Feld soll der Logical ID des ausgewählten Objekts eingetragen werden. Dieser Logical ID dient zur eindeutigen Identifizierung des Objekts. Aus diesem Grund soll ein Logical ID nur einmal vorkommen. Obwohl dieser Logical ID für das Netzmanagement irrelevant ist, erleichtert er das Erkennen der ausgewählten Objekte, insbesondere wenn sie mehrmals ausgewählt worden sind. Als Default-Wert für den Logical ID erscheint am Anfang automatisch der Physical ID.

3. `Kind`: In dieses Feld soll eingetragen werden, was das ausgewählte Objekt präsentiert. Die Liste der Möglichkeiten wird bei der Initialisierung des Konfigurators aus einer Initialisierungsdatei gelesen und hier automatisch zur Verfügung gestellt.

4. `Description`: Dieses Feld beinhaltet eine ausführlichere Beschreibung des Objekts.

5. `Type`: Jedes Objekt liefert einen Wert. In diesem Feld soll der Datentyp dieses Feldes stehen. Wie oben schon erwähnt, werden die mögliche Datentypen bei der Initialisierungsphase des Konfigurators aus einer Initialisierungsdatei gelesen und hier zum Auswahl angeboten.

6. `Index`: Dieses Feld ist nur für die Arrays relevant. Hier steht die niedrigste und höchste Indexnummer eines Arrays. Falls das Objekt kein Array ist, erscheint der Text "This is not an array!".

7. `Place`: Dieses Feld dient zur Eingabe eines speziellen Codes für den Platz des jeweiligen Knotens mit dem ausgewählten Objekt.

Die hier eingetragenen Daten werden dann in eine Konfigurationdatei gespeichert und nachher aus dieser Datei vom P-NET Sub-Agent gelesen. Nachdem sich ein Master-Agent bei dem Sub-Agent anmeldet, werden diese Daten vom Sub- zum Master-Agent gesendet, der sie dann in seine MIB einträgt.

Man kann diese Daten, außer `Physical ID` und `Logical ID,` auch während des Auswählens einfach weglassen, und später eintragen. Entweder durch das Deaktivieren der Menüfunktion "`Ask properties by each selection`", die unter dem Menü "`View`" zu finden ist, oder durch das Aktivieren des Kontrollkästchens "`Do not show this dialog next time`" auf dem Dialogfenster, kann das Erscheinen des Dialogfensters bis zur nächsten Aktivierung dieser Menüfunktion unterbunden werden. Da werden die `Logical ID` automatisch dem `Physical ID` gleichgesetzt und die anderen Felder leer gelassen. Nur wenn man ein Objekt zweimal auswählen will, wird das obengenannte Dialogfenster aufgerufen, da jede `Logical ID` nur einmal vorkommen soll. Da diese Eigenschaften der Objekte für das Netzmanagement von eminenter Wichtigkeit sein können, ist es natürlich nicht ratsam, sie für immer auszulassen und dann die Konfigurationsdatei in ihrer Unvollkommenheit vom Sub-Agent verwenden zu lassen.

Die Eigenschaften der ausgewählten Objekte können jederzeit modifiziert werden, indem man zuerst das Objekt aus dem Fenster "`selected Objects`" bzw. "`selected Logical ID.s`" auswählt und dann entweder die Tastenkombination `ALT+P` drückt oder die Funktion "`Properierties`" unter dem Menü "`View`" aufruft.

6. SUB-AGENT FÜR P-NET

6.1 Allgemeine Eigenschaften

Während dieser Diplomarbeit wurde ein Sub-Agent für das Feldbussystem P-NET programmiert, was eigentlich die Hauptaufgabe dieser Diplomarbeit ausmacht.

Die Spezifikation für einen allgemeinen Sub-Agent wurde bereits im Kapitel 5 ausführlich dargelegt. In diesem Kapitel wird die Realisierung dieser Kriterien in den Sub-Agent für P-NET geschildert.

Wie bereits im Kapitel 4.5 erläutert wurde, können die Sub-Agents in zwei Gruppen unterteilt werden, in aktive und passive Sub-Agents. Der Sub-Agent für P-NET ist als ein passiver Sub-Agent konzipiert worden, der im Gegensatz zu aktiven Sub-Agents keine Möglichkeit bietet, über den Master-Agent auf andere Sub-Agenten und somit auf andere Feldbussysteme zuzugreifen. Infolgedessen besitzt er nur ein beschränktes Maß an Intelligenz und überläßt alle managementrelevanten Entscheidungen dem Master-Agent.

Der Sub-Agent ist allgemein für den Zugriff auf das Feldbussystem verantwortlich. Der Sub-Agent für P-NET verwendet das Feldbus-Management-System VIGO als eine standardisierte Benutzerschnittstelle für den Zugriff auf den Feldbus, welches im Kapitel 3.6 erklärt wurde. VIGO unterstützt derzeit nur das Feldbussystem P-NET, aber im Rahmen eines Entwicklungsprojekts der Herstellerfirma PROCES-DATA AS wird es versucht, die Verwaltung anderer Feldbussysteme, wie Profibus oder WorldFIP, durch VIGO zu verwirklichen. Diese Möglichkeit erhöht die Anforderungen für den Sub-Agent, der so flexibel entwickelt werden soll, daß eine solche Erweiterung in VIGO ohne großen Aufwand in den Sub-Agent integriert werden kann. Damit kann der Sub-Agent nicht nur P-NET sondern auch die anderen Feldbussysteme betreuen. Tatsächlich braucht man für eine Erweiterung nur die alte Schnittstelle zwischen dem Sub-Agent und VIGO mit der von der Herstellerfirma gelieferten neuen Schnittstelle zu ersetzen und danach das Programm neu zu kompilieren, was man innerhalb ein paar Minuten erledigen kann.

Der Sub-Agent wurde, wie der Konfigurator, mit der Programmiersprache C++ unter der Verwendung der Entwicklungsumgebung MS Visual C++ 5.0 geschrieben. Die ausführbare Datei heißt "pna.exe". Der Sub-Agent für P-NET ist eine 32-Bit-Applikation, die nur auf einem PC mit einem Betriebsystem MS Windows 9x/NT lauffähig ist. Diese Einschränkung wurde durch die Benutzung von VIGO und durch die systemabhängige Multhithread-Fähigkeit des Sub-Agents leider unvermeidlich.

Das Programm besitzt eine einfache graphische Benutzeroberfläche (siehe Abb. 6.1). Sie besteht hauptsächlich aus zwei Teilen:

1. Ein Statusfenster, das zur Anzeige der wichtigen Ereignisse, wie die Verbindungen mit den Master-Agents oder Fehlermeldungen, dient.

2. Ein Menüteil mit diversen Funktionen und zwei Schaltflächen "`Open`" und "`Exit`".

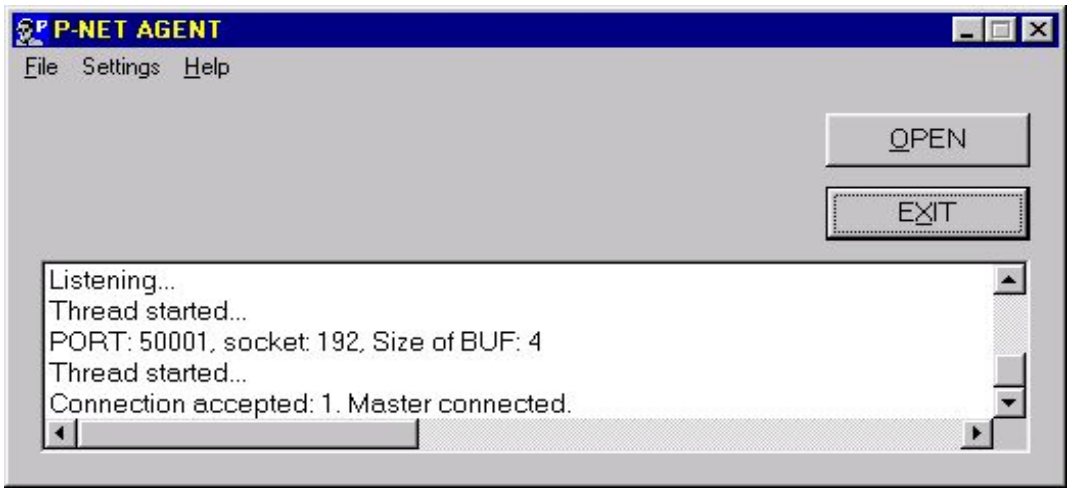


Abb. 6.1 Benutzeroberfläche des Sub-Agents

Die Schaltfläche "Open" dient dazu, das Programm neu zu starten, wobei man auch eine neue Initialisierungsdatei angeben kann. Dabei werden alle Verbindungen mit den Master-Agents unterbrochen, und das Programm initialisiert sich neu. Diese Initialisierungsphase wird in dem folgenden Kapitel ausführlich erläutert.

6.2 Initialisierungsphase des Sub-Agents

Der Programmablauf des Sub-Agents kann grundsätzlich in zwei Teile gegliedert werden:

1. Initialisierungsphase

2. Kommunikation mit dem Master-Agent

Die Initialisierungsphase kann man als den Abschnitt vom Aufrufen des Programms bis zum Warten auf eine Verbindung mit einem Master-Agent auffassen.

Nach dem Start sucht der Sub-Agent als erstes nach einer Initialisierungsdatei. Der Sub-Agent kann entweder mit einem oder ohne Parameter aufrufen werden. Durch diesen Parameter wird eine Initialisierungsdatei angegeben, wonach sich der Sub-Agent richten soll. Damit wurde die

Flexibilität des Programms erhöht, da man je nach Bedarf verschiedene Initialisierungsdateien benutzen kann. Die Syntax des Sub-Agents sieht dann folgendermaßen aus:

```
pna.exe [initialization_file]
```

Falls kein Parameter eingegeben wird, wird die Default-Datei "pna.ini", die in dem selben Verzeichnis wie das Programm "pna.exe" sein muß, als Initialisierungsdatei benutzt. Falls diese Datei auch nicht existiert, beendet das Programm nach einer Fehlermeldung.

Die Initialisierungsdatei besteht aus zwei Teilen:

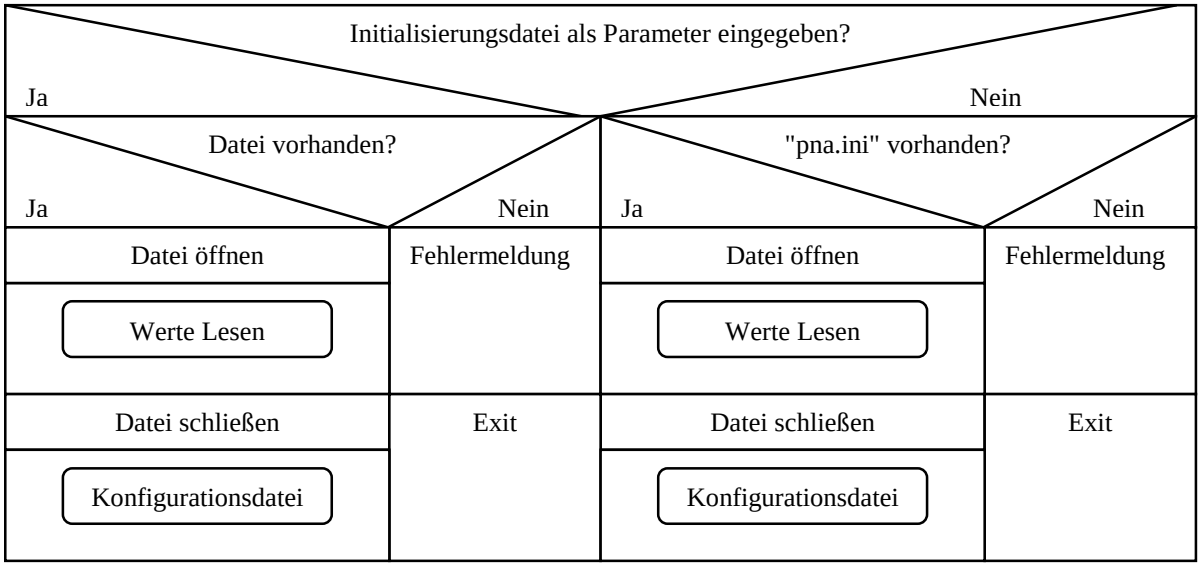
1. Der allgemeine Konfigurationsteil, der unter dem Abschnitt "`Configuration`" zu finden ist und die Konfigurationsdatei und Portnummern, die bei der Sub-Agent Registration (SAR) und Kommunikation mit Master-Agents verwendet werden. Die Portnummer für die Kommunikation mit Master-Agents ist unter dem Schlüsselwort "`portNo2`" zu finden, während die für die SAR mit dem Schlüsselwort "`portNo`" registriert sein soll.

2. Der Abschnitt mit dem Schlüsselwort "`Masters`", der die URL-Adresse und Portnummer der bekannten Master-Agents beinhaltet, bei denen sich der Sub-Agent infolge der SAR anmelden soll.

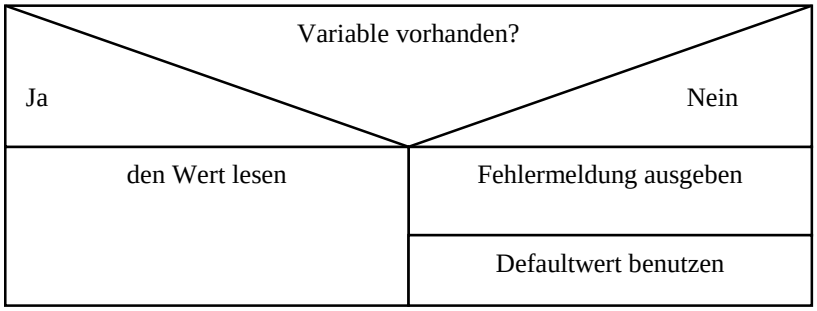
Für den ersten Teil hat jede Variable auch einen Default-Wert, der bei Nichtexistenz des jeweiligen Wertes in die Initialisierungsdatei eingesetzt wird. Zum Beispiel wurde für die Konfigurationsdatei, die mit Hilfe des Konfigurators erstellt wurde, die Datei "pna.cfg", die in dem selben Verzeichnis wie das Programm "pna.exe" vorhanden sein muß, als Default-Wert festgelegt. Falls keine Angaben über die Konfigurationsdatei zu finden sind und diese Default-Datei nicht existiert, dann terminiert das Programm mit einer Fehlermeldung.

Der zweite Teil enthält die Internetadressen der Master-Agents zusammen mit ihren jeweiligen Portnummern. Diese Angaben benötigt der Sub-Agent zum Zwecke der SAR, wobei er bei den jeweiligen Master-Agents angibt, wo er auf eine Verbindung mit ihnenwartet.

Initialisierung:



Werte Lesen:



Konfigurationsdatei:

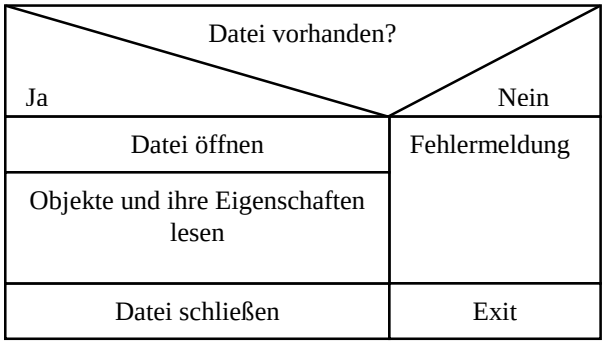


Abb. 6.2 Struktrogramm für die Initialisierungsphase des Sub-Agents

Nachdem die Daten in der Initialisierungsdatei verarbeitet wurden, wird die Konfigurationsdatei, die mit Hilfe des Konfigurators erstellt wurde, gelesen. In der

Konfigurationsdatei stehen die Objekte mit ihrer Eigenschaften in einer vorgegebenen Reihenfolge. Diese Reihenfolge bestimmt auch die ID-Nummer der verwalteten Objekte, die durchgehend von 1 bis n sein sollen, wobei n die Gesamtanzahl der Objekte ist, und von dem Sub-Agent und den Master-Agents zur Identifizierung des jeweiligen Objekts benutzt werden. Außer dem sogenannten Physical ID jedes Objekts, mit dem der Sub-Agent über VIGO auf das jeweilige Objekt zugreifen kann, stehen die logische Adresse (Logical ID), Objektart, eine textuelle Erklärung des Objekts, der Datentyp des Wertes des Objekts und eine Platzbeschreibung des Objekt zeilenweise in der geschriebenen Reihenfolge. Diese Eigenschaften werden mit dem Zeichen "--" am Anfang jeder Zeile gekennzeichnet. Für jedes Objekt wurde ein VIGO-Objekt definiert, über das der Sub-Agent auf das physikalische Objekt zugreifen kann, und dann intern in ein Array gespeichert. Die Abbildung 7.2 zeigt ein vereinfachtes Struktogramm für die Initialisierungsphase des Sub-Agents.

Im Kapitel 4.5 wurde erläutert, wie die verwalteten Objekte auf dem Sub-Agent allgemein gespeichert werden sollen, damit ein hoher Speicherbedarf vermieden werden kann. Dabei wurde in Betracht gezogen, daß zusammengesetzte Objekte wie zum Beispiel Bitarrays mehrmals ausgewählt werden würden, weil man für ein bestimmtes Bit den Wert des ganzen Arrays abfragen sollte. Da aber in VIGO eine derartige Abfrage ausgeschlossen ist und man AUF jedes einzelne Bit abgesondert zugreifen muß, ist die vorgeschlagene Speicherung in zwei Arrays nicht mehr vorteilhaft. Deswegen wurden alle verwalteten Objekte zusammen mit ihren zugehörigen VIGO-Objekten in ein einziges Array gespeichert. Die mehrfach ausgewählten Objekte werden dann getrennt voneinander verwaltet, was den benötigten Speicher, aber auch die Geschwindigkeit des Programms erhöhen wird. Der zusätzliche Speicherbedarf wird diesmal nicht so groß sein wie bei der vorherigen Überlegung angenommen wurde, da die Anzahl der vielfach ausgewählten Objekte gering sein wird, weil man einen Bitarray nicht mehr mehrfach auswählt, sondern jedes Element dieses Arrays einzeln abgelegt wird.

Die Abbildung 6.3 zeigt die Speicherung der verwalteten Objekte in ein Array. Jedes Objekt enthält außer seinen Eigenschaften ein virtuelles VIGO-Objekt, mit dem ein Zugriff auf das jeweilige physikalische Objekt über VIGO ermöglicht wird. Wie dieses virtuelle Objekt definiert werden kann, wurde schon in dem Kapitel 3.6.1 dargestellt. Das gespeicherte Objekt, das im Folgenden als SA-Objekt bezeichnet wird, enthält außer seinen Eigenschaften zwei weitere Variablen, eine Variable für die Speicherung des Wertes des jeweiligen Objekts und eine andere für die Festhaltung des Zeitpunktes des Wertempfanges. Bei einer Wertabfrage dient diese zweite Variable dazu, um festzustellen, ob der gespeicherte Wert gemäß der in der Master-Agent-Abfrage definierten Zeitspanne, die das maximale Alter des gespeicherten Wertes vorgibt, aktuell genug ist oder nicht. Falls ja, wird dieser zum Master-Agent zurückgeliefert.

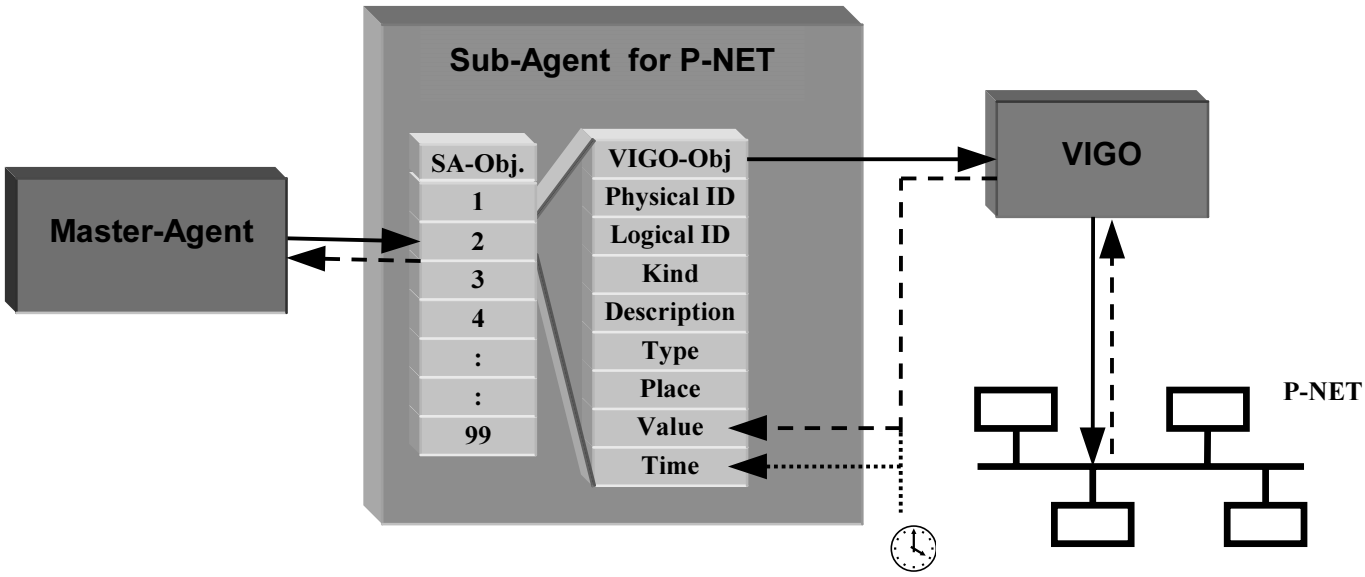


Abb. 6.3 Speicherung der SA-Objekte

6.3 Multithreading im Sub-Agent

Nachdem alle Daten aus der Konfigurationsdatei gelesen wurden, fängt der Sub-Agent mit den Vorbereitungen für die Verbindung mit Master-Agents an. Der Sub-Agent von P-NET ist multimasterfähig, d.h. mehrere Master-Agent können von dem Sub-Agent gleichzeitig betreut werden. Um eine möglichst parallele Verwaltung der Verbindungen mit den Master-Agents bewerkstelligen zu können, ist der Sub-Agent als eine Applikation mit Multithreading-Fähigkeit implementiert worden. Ein einzelner Prozeß kann verschiedene Ausführungspfade (Threads) enthalten, die vom Betriebssystem verwaltet werden und über eigene Stacks verfügen, aber den gesamten Adreßraum mit Code- und Datenbereich benutzen dürfen. Der Sub-Agent verwendet für jede Verbindung mit einem Master-Agent einen separaten Thread, für den die globalen Daten der Sub-Agent-Applikation so wie das Array mit den SA-Objekten zugänglich sind (siehe Abb. 6.4). Es existiert außerdem immer ein Thread, der auf eine neue Verbindung wartet.

Nachdem die Konfigurationsdatei gelesen wurde, wird sofort ein neuer Thread gestartet, der einen Socket entsprechend der Socketnummer, die in der Initialisierungsdatei unter dem Schlüsselwort "`portNo2`" zu finden ist, aufbaut. Der Thread geht danach in den Wartezustand, in dem er solange bleibt, bis eine Verbindung mit einem Master-Agent zustande kommt. Gleichzeitig wird ein neuer Thread gestartet, der wiederum auf eine weitere Verbindung mit einem Master-Agent wartet.

Die Abbildung 6.5 zeigt ein Struktrogramm, welches die notwendigen Schritte für den Aufbau einer Socket-Verbindung in einem vereinfachten Form darstellt. Um bei einem Vergleich mit dem Sourcecode das Verständis der Abläufe zu erhöhen, werden die für den Verbindungsaufbau relevantenVisual-C++-Befehle *kursiv* dazu geschrieben, obwohl das in einem Struktrogramm nicht üblich ist.

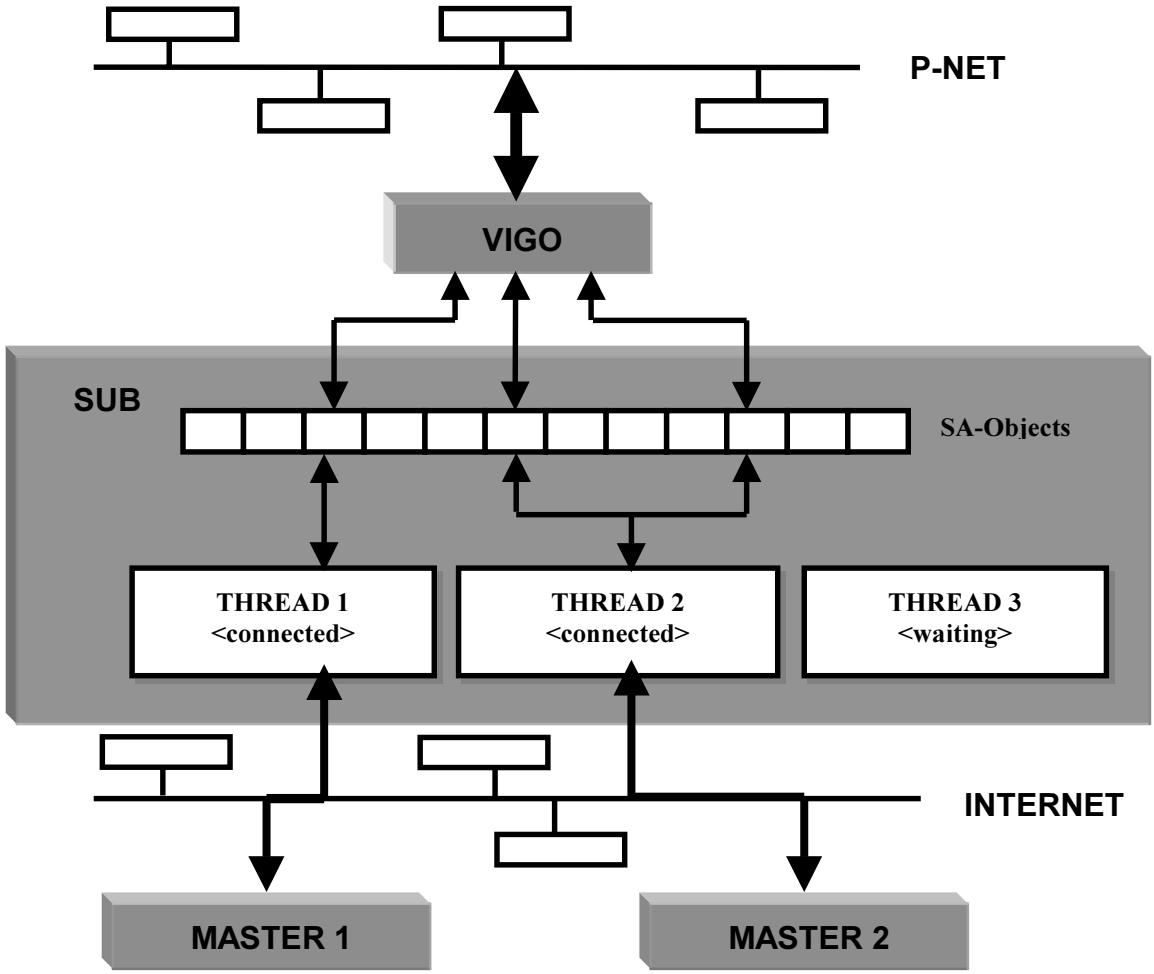


Abb. 6.4 Multithreading im Sub-Agent

Da auch VIGO selbst eine parallele Durchführung mehrerer voneinander unabhängiger Buszugriffe erlaubt, wird eine gleichzeitige Ausführung mehrerer Threads, die gemäß der Kommunikation mit dem jeweiligen Master-Agent auf das Feldbussystem zugreifen, kein Problem verursachen. Nur ein gleichzeitiger Schreibzugriff auf das gleiche Objekt durch mehrere Master-Agents sollte vermieden werden, um Inkonsistenzen und andere mögliche Kollisionen zu vermeiden. Diese Buszugriffe, in denen der Wert eines Objekts gesetzt wird, bilden somit den kritischen Abschnitt der Kommunikation und sollen folglich synchronisiert werden. Unter der Verwendung des Synchronisationswerkzeugs "`CriticalSection`" wird einem Thread, und dadurch einem Master-Agent, ein exklusiver Zugriffsrecht auf ein bestimmtes Objekt erteilt. Solange ein Thread den Wert eines Objekts liest oder einsetzt, können andere Threads nicht darauf zugreifen.

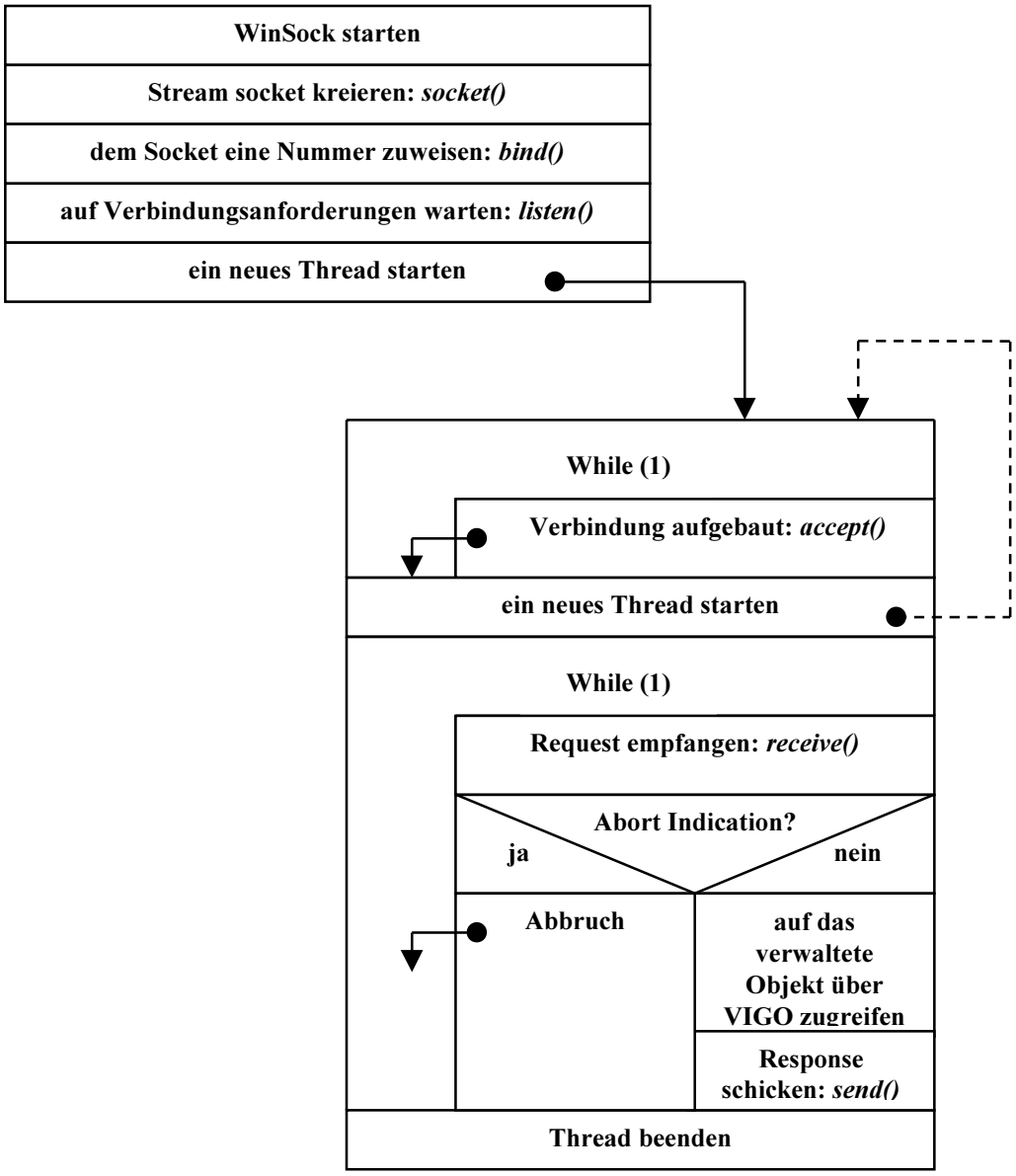


Abb. 6.5 Struktrogramm für den Verbindungsaufbau und Kommunikation

Es wurde keine Obergrenze für die Anzahl der Master-Agents vorgegeben, obwohl eine hohe Anzahl der Master-Agents zu Engpässen führen kann, insbesondere wenn mehrere Master-Agents auf die selben Feldbus-Objekte zugreifen wollen. Außerdem können die Betriebsysteme Windows 9x/NT bis auf eine beschränkte Anzahl von Threads problemlos unterstützen, die auch von dem benutzten Computer abhängt. Aber es wurde vorerst angenommen, daß eine derartig große Anzahl der an den Sub-Agent angeschlossenen Master-Agents sehr unwahrscheinlich wäre. Deshalb wurde eine Begrenzung der Anzahl der möglichen Verbindungen mit den Master-Agents vorerst nicht implementiert.

6.4 Kommunikation mit Master-Agents

Nach der Initialisierung existiert immer ein Thread, der auf eine Verbindung mit einem Master-Agent wartet. Wenn ein Master-Agent sich an den Sub-Agent über Internet anschließen will, soll er die Internet-Adresse (URL) des Sub-Agents kennen. Normalerweise wird diese während der Sub-Agent Registration (SAR) dem Master-Agent bekanntgegeben. Der Master-Agent soll danach eine Socket-Verbindung mit dem jeweiligen Port auf dieser Adresse aufbauen, auf dem der Sub-Agent für Master-Agents ansprechbar ist. Der Master-Agent schickt dann die Meldung "`ConnectionRequest`" zum Sub-Agent. Falls der Sub-Agent diese Verbindung akzeptiert, sendet er die Meldung "`ConnectionResponse`" mit einem Fehlercode Null.

An dieser Stelle wird eine Sicherheitslücke des Sub-Agents bemerkbar. In dem Sub-Agent wurde keine Maßnahmen für eine Authentifizierungs- und Berechtigungsprüfung der Master-Agents gesetzt, die um eine Verbindung mit dem Sub-Agent ansuchen. Jeder Master-Agent kann sich beim Sub-Agent anmelden und dann über ihn auf das Feldbussystem zugreifen. Das könnte unerwünschte Folgen mit sich bringen, da über einem Manager oder Master-Agent Daten aus dem Feldbus gelesen oder auch geändert werden können. Obwohl eine einfache Beschränkung der ansprechbaren Master-Agents ganz leicht implementiert werden kann, braucht das ganze WEBFAN-Projekt, dem auch diese Diplomarbeit auch gehört, ein komplexeres Scherheitssystem, mit dem die Zugriffe auf die Feldbussysteme beschränkt oder total verweigert werden. Da der Entwurf eines solchen Systems, was sehr kompliziert und aufwendig sein kann, den Aufgabenbereich dieser Diplomarbeit übersteigt, gehört es nicht zu den Aufgaben dieser Arbeit. Da das Projekt WEBFAN noch immer in der Planungsphase ist, wurde die Implementierung eines Sicherheitssystems auf spätere Zeit verschoben. Deswegen wurden die für eine ernsthafte Verwendung unseres Netzmanagementsystem benötigten Maßnahmen bezüglich der Datensicherheit nicht implementiert.

Nachdem eine Verbindung mit einem Master-Agent aufgebaut worden ist, kann der Master-Agent über das Protokoll, das im Kapitel 4.7 beschrieben wurde, mit dem Sub-Agent kommunizieren und somit auf das Feldbussystem zugreifen. Hier verbirgt sich noch ein kleiner Mangel: Die Kommunikation zwischen Master- und Sub-Agent basiert auf einen Protokoll, der noch nicht vollkommen konzipiert worden ist. Wie schon erwähnt, ist diese Diplomarbeit ein Teil eines großen Projekts, namens WEBFAN, dessen Struktur noch in Entwicklung ist. Deswegen könnte das Protokoll im Laufe dieser Entwicklungsphase modifiziert werden. Da könnte der Master-Agent eine andere Version des Protokolls benutzen als der Sub-Agent selbst. Alle Anfragen des Master-Agents, die der Protokollversion des Sub-Agents nicht entsprechen werden, werden einfach ignoriert. Obwohl eine unkomplizierte Versionsabfrage des verwendeten Protokoll integriert werden könnte, um herauszufinden, ob die Versionen aller Kommunikationsteilnehmern miteinander kompatibel sind oder nicht, wurde eine solche Abfrage bis zur Fertigstellung dieser Diplomarbeit im Protokoll nicht vorgesehen.

Außerdem müssen alle Nachrichten von Master-Agents gemäß dem Protokoll und den ASN.1 Regeln richtig codiert werden, ansonsten werden sie einfach vom Sub-Agent ignoriert.

Ein anderer wichtiger Punkt bei der Kommunikation zwischen dem Master- und Sub-Agent ist, daß die Codierung der Datentypen bei allen Kommunikationspartnern übereinstimmen muß. Weil nicht die Datentypen der verwalteten Objekte selbst, sondern nur ihre Codenummern den Master-Agents bekanntgegeben werden, ist es insbesondere für die Master-Agents, die die Werte der verwalteten Objekte nicht nur lesen sondern auch modifizieren wollen, erforderlich, dieselben Nummer für die Datentypen in ihrer "`SetRequest`"-Anfrage zu benutzen. Sonst kann eine Modifikation des jeweiligen Wertes nicht erfolgen, und der Master-Agent bekommt eine "`SetResponse`"-Meldung mit einem entsprechenden Errorcode (ungleich Null) zurück.

Bevor die Sub-Agent-Applikation beendet wird, werden zuerst alle Verbindungen mit den Master-Agents getrennt. Da wird jedem Master-Agent die Nachricht "AbortIndication" geschickt und der Sub-Agent wartet, bis er eine Rückmeldung von den Master-Agents, enbenfalls die Meldung "AbortIndication", zurückbekommt. Wenn aber die Rückmeldung vom Master-Agent innerhalb einer bestimmten Wartezeit nicht kommt, wird die Trennung der Verbindung trotzdem durchgeführt, um zum Beispiel bei einem Absturz des Master-Agents ein unendliches Warten zu vermeiden.

6.5 Testprogramm für den Sub-Agent

Da während der Ausarbeitung dieser Diplomarbeit noch kein Master-Agent existierte, wurde auch ein Testprogramm für die Sub-Agent-Applikation geschrieben (siehe Abb. 6.6), das alle möglichen Funktionen eines Master-Agents simuliert, die für die Kommunikation mit dem Sub-Agent für P-NET relevant sind.

Auf der Benutzeroberfläche dieses Programms findet man zwei Eingabefelder, nämlich "`URL`" und "`PORT`", in die man die IP-Adresse und Portnummer eingeben muß, unter dem der Sub-Agent ansprechbar ist. Durch die Betätigung des Schaltknopfes "`Connect`" kann das Testprogramm mit dem Sub-Agent verbunden werden. Nach einer erfolgreichen Initialisierungsphase werden die ID-Nummern der verwalteten Objekte in der Liste "Objekt IDs" erscheinen. Wenn man ein Objekt aus dieser Liste auswählt, werden im Teil "`Properties`" die Eigenschaften dieses Objekts sichtbar. Im Feld "`Place`" steht nur die Meldung "`not used`", weil die Übertragung der Platzinformationen im Kommunikationsprotokoll noch nicht vorgesehen ist, obwohl dieses Feld im Sub-Agent und Konfigurator schon integriert ist.

Mit folgenden Schaltflächen werden verschiedene Funktionen eines Master-Agents simuliert:

- `Get`: Mit der Betätigung dieses Schaltknopfs kann eine `GetRequest`-Meldung an den Sub-Agent geschickt werden, um den Wert des Objekts aus dem Feldbussystem zu holen. Da muß die sogenannte Delta-Zeit angegeben werden, die besagt, wie alt ein im Puffer des Sub-Agenten gespeicherten Wert höchstens sein kann, damit dieser zum Master-Agenten gesendet werden kann, ohne auf den Feldbus zuzugreifen. Die Informationen aus der jeweiligen Rückmeldung

"`GetResponse`" werden in den entsprechenden Feldern präsentiert.

- `Get All`: Diese Schaltfläche dient, um den Wert jedes verwalteten Objekts zu erhalten, wobei für jedes Objekt ein "`GetRequest`" geschickt wurde.

- `Set`: Mit dem Schaltknopf "`Set`" wird eine `SetRequest`-Meldung zum Sub-Agent gesendet. Das `SetRequest`-Befehl kann den Wert eines verwalteten Objekts modifizieren. Da aber das Testprogramm so einfach wie möglich aufgebaut ist, wird die Aufgabe, einen neuen Wert des richtigen Datentyps auszuwählen, dem Benutzer dieses Testprogramms überlassen. Deshalb soll der Benutzer aufpassen, daß der neue Wert dem Datentyp des Objekts entspricht. Falls der neue Wert nicht erfolgreich gesetzt werden kann, wird der Fehlercode, der ungleich Null sein wird, in dem dazugehörigen Fenster mit der Bezeichnung "`Error`" erscheinen.

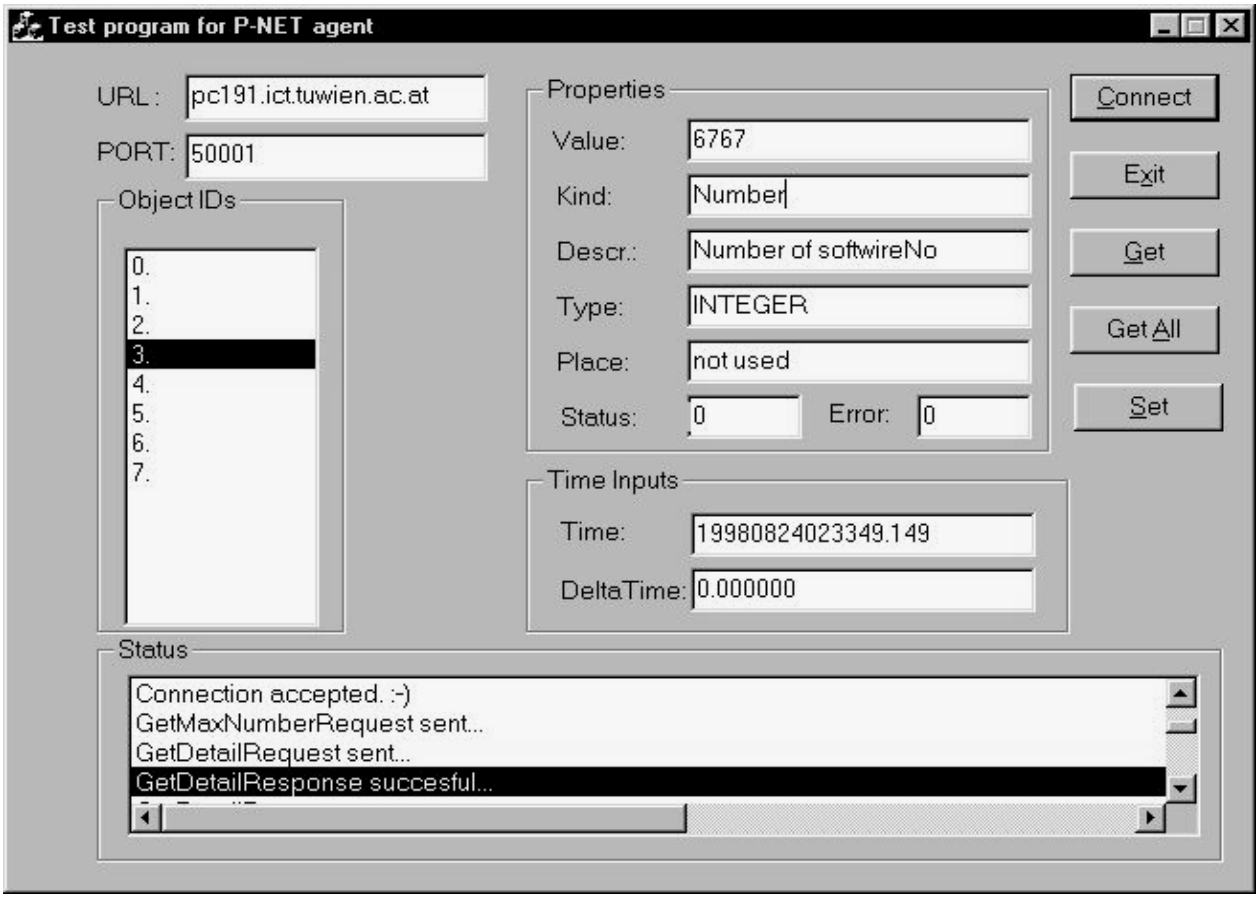


Abb. 6.6 Testprogramm für den Sub-Agent

Ferner gibt es auf der Benutzeroberfläche des Testprogramms ein weiteres Feld, das aus zwei Fenstern für die zeitrelevanten Daten besteht. Im Fenster "`Time`" erscheint die Zeit in dem ASN.1-Datentyp "`TimeTicks`", die aus einem Response vom Sub-Agent erhalten wurde. Im anderen Fenster "Delta Time" soll, wie oben beschrieben, bei einer Get- oder GetAll-Funktion die Delta-Zeit angegeben werden. Wenn eine Response-Meldung vom Sub-Agent kommt, wird hier die Dauer der Durchführung der jeweiligen Request-Anfrage beim Sub-Agent angezeigt.

Das Testprogramm ist bewußt einfach programmiert worden. Da das Programm nur für Testzwecke dient, wurde nur das Nötigste implementiert. Darum kann zum Beispiel keine zweite Verbindung mit dem Sub-Agent aufgebaut werden, wenn bereits eine Verbindung schon existiert. In diesem Fall muß das Programm einfach noch einmal wieder neu gestartet werden.

7. RESÜMEE

Diese Diplomarbeit ist ein Teil eines Projekts, namens "WEBFAN", am Institut für Computertechnik der Technischen Universität Wien. In diesem Projekt wird versucht, ein flexibles Netzmanagementsystem zu konzipieren, mit dem verschiedene Feldbussysteme wie Profibus oder P-NET, über Internet unter Benutzung von SNMP verwaltet werden können, wobei in dieser Diplomarbeit die Anbindung des Feldbussystems P-NET an dieses Managementsystem implementiert wurde.

Durch den modulare Aufbau des Agents kann das entworfene System auf weitere Feldbussysteme erweitert werden. Da in nächster Zeit neben P-NET auch andere Feldbussysteme wie Profibus von dem, bei der Implementation des Sub-Agents verwendeten, Feldbus-Management-System VIGO verwaltet werden können, könnte der hier erarbeitete Sub-Agent auch für das Management dieser Feldbusse eingesetzt werden.

Während der Bearbeitung dieser Diplomarbeit war die Planungsphase des Projekts noch nicht abgeschlossen und diese erlebte ständig verschiedene Modifikationen, die natürlich nicht unerhebliche Wirkungen auf diese Arbeit hatten. Diese Diplomarbeit mit den dazugehörigen Programmen entspricht in ihrem letzten Zustand, den für das Projekt vorgesehenen Erfordernissen. Bei der Programmierung der Applikationen wurde versucht, sie möglichst flexibel und modular zu gestalten, damit zukünftige Änderungen ohne großen Aufwand implementiert werden können.

Das Kommunikationsprotokoll zwischen dem Master- und Sub-Agent ist ein Teil des Projekts, welches in nächster Zeit mit vielen Erweiterungen konfrontiert werden wird. Obwohl viele von den zukünftigen Änderungen schon besprochen worden sind, wurden sie bis zu dem Zeitpunkt der Abschließung dieser Arbeit nicht in das Protokoll integriert. Folglich kann der Sub-Agent nur einen beschränkten Teil des zukünftigen Protokolls verwenden.

In nächster Zeit werden höchstwahrscheinlich andere Erweiterungen im Projekt, insbesondere bezüglich der Sicherheit, vorgenommen werden. Im Projekt "WEBFAN" wurde das Thema "Sicherheit" momentan außer acht gelassen, da man sich zuerst auf die genauere Analyse der allgemeinen Struktur des Systems konzentrieren wollte. Nach genauer Untersuchung der Interaktionen der Elemente des Systems und Festlegung mögliche Einsatzbereiche dieses Managementsystems können die erforderlichen Kriterien für die Sicherheit bestimmt werden. Danach können auch die notwendigen Maßnahmen getroffen werden.

Für die Kommunikation zwischen dem Master-Agent und Manager wurde zum Beispiel die erste Version von SNMP (SNMPv1) trotz mangelnder Sicherheit ausgewählt, weil diese weit verbreitet ist. Falls später das ganze System für sicherheitskritische Anwendungen verwendet werden soll, könnte die jetzige Version von SNMP durch die letzte (SNMPv3) ersetzt werden, die die neue Erweiterungen im Bezug auf die Sicherheit und Administration zur Verfügung stellt.

Aus dem oben genannten Grund wurden in der Sub-Agent-Applikation auch die Maßnahmen, die für die Authentifizierung der Master-Agents und für die Erteilung der Zugriffsrechte erforderlich sind, nicht ergriffen. Daher kann nicht festgestellt werden, ob es sich um einen zulässigen Master-Agent handelt, der über dem Sub-Agent auf das Feldbussystem zugreift, oder nicht.

Nennenswert ist ein weiterer Schwachpunkt der Applikationen und zwar ihre Abhängigkeit von dem Betriebssystem. Aus diesem Grund, daß im Konfigurator und Sub-Agent benutzten VIGO eine 32-Bit-Windows-Applikation ist und die Multitasking-Fahigkeit des Sub-Agents unter Benutzung von betriebssystemspezifischen Funktionen realisiert werden kann, können die in dieser Diplomarbeit entworfenen Applikationen nur unter dem Betriebssystemen Windows 9x/NT laufen.

Zum Schluß ist es erwähnenswert, daß das ganze System nicht für zeitkritische Anwendungen konzipiert ist, weil das Internetprotokoll (IP), auf dem unser System basiert, nicht dazu fähig ist. Andererseits kann die Schnittstelle zwischen dem Feldbus und Sub-Agent die möglicherweise auftretende Informationsmenge nicht bearbeiten.

LITERATURVERZEICHNIS

[FEI95] Feit, Sidnie M.: SNMP : a guide to network management. - New York, NY [u.a.] :McGraw-Hill, 1995.
ISBN 0-07-020359-8.

[GIL95] Andreas Gillhuber, "Management im Netz", MC Extra, 11/95.

[GOR92] Walter Gora: ASN.1, Abstract Syntax Notation One. Datacom-Verlag, 1992.

[KKM97] Martin Knizak, Michael Kunes, Martin Manninger, Thilo Sauter: Modular Agent Design for Fieldbus Management

[MIL93] Mark A. Miller: Managing internetworks with SNMP : the definitive guide to the simple network management protocol (SNMP) and SNMP version 2. - New York, NY : M & T Books, 1993.
ISBN 1-55851-304-3.

[NET96] International P-NET User Organization: P-NET Booklet. - 1996.

[PER93] David T. Perkins: Understanding SNMP MIBs. - Revision 1.1.7, 1993.

[PPM91] PROCES-DATA A/S: Process Pascal V2.0 Users Manual. - 1991.

[RFC768] Request for Comments Dokument Nr.768: User Datagram Protocol.

[RFC791] Request for Comments Dokument Nr.791: Internet Protocol.

[RFC793] Request for Comments Dokument Nr.793: Transmission Control Protocol.

[RFC1141] Request for Comments Dokument Nr.1141: Introduction to version 2 of the Internet-standard Network Management Framework.

[RFC1155] Request for Comments Dokument Nr.1155: Structure and Identification of Management Information for TCP/IP-based Internets.

[RFC1156] Request for Comments Dokument Nr.1156: Management Information Base for Network Management of TCP/IP-based internets.

[RFC1227] Request for Comments Dokument Nr.1227: SNMP MUX Protocol and MIB.

[RFC1228] Request for Comments Dokument Nr.1228: SNMP-DPI, Simple Network Management Protocol Distributed Program Interface.

[RFC1901] Request for Comments Dokument Nr.1901: Introduction to Community-based SNMPv2.

[RFC1902] Request for Comments Dokument Nr.1902: Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2).

[RFC1903] Request for Comments Dokument Nr.1903: Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2).

[RFC1904] Request for Comments Dokument Nr.1904: Conformance Statements for Version 2 of the Simple Network Management Protocol (SNMPv2).

[RFC1905] Request for Comments Dokument Nr.1905: Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2).

[RFC1906] Request for Comments Dokument Nr.1906: Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2).

[RFC1907] Request for Comments Dokument Nr.1907: Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2).

[RFC1909] Request for Comments Dokument Nr.1908: An Administrative Infrastructure for SNMPv2.

[RFC2011] Request for Comments Dokument Nr.2011: SNMPv2 Management Information Base for the Internet Protocol using SMIv2.

[RFC2012] Request for Comments Dokument Nr.2012: SNMPv2 Management Information Base for the Transmission Control Protocol using SMIv2 .

[RFC2013] Request for Comments Dokument Nr.2013: SNMPv2 Management Information Base for the User Datagram Protocol using SMIv2.

[RFC2257] Request for Comments Dokument Nr.2257: Agent Extensibility (AgentX) Protocol.

[RFC2271] Request for Comments Dokument Nr.2257: An Architecture for Describing SNMP Management Frameworks.

[RFC 2573] Request for Comments Dokument Nr.2573: SNMPv3 Applications.

[RFC 2574] Request for Comments Dokument Nr.2574: User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3).

[RFC 2575] Request for Comments Dokument Nr.2575: View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP).

[ROS93] Rose, Marshall T.: Einführung in die Verwaltung von TCP-IP-Netzen : Netzwerkverwaltung und das Simple Network Management Protocol (SNMP) . Aus dem Engl. übers. v. Hardy Dölfel. - München; Wien : Hanser [u.a.], 1993.
ISBN 3-446-16444-8 - ISBN 0-13-812959-2 - ISBN 3-446-16444-0.

[ROS95] Rose, Marshall T. ; Keith McCloghrie: How to manage your network using SNMP : the networking management practicum. - Englewood Cliffs, NJ : Prentice-Hall, 1995.
ISBN 0-13-141517-4.

[STA93] Stallings, William: SNMP, SNMPv2, and CMIP : the practical guide to network-management standards. - 3. print. - Reading, Mass. [u.a.] : Addison-Wesley, 1993.
ISBN 0-201-63331-0.

[STA98] Stallings, William: SNMPv3: A Security Enhancement to SNMP: IEEECommunications Surveys, Fourth Quarter 1998, Vol. 1 No. 1 17.

[VUM97] PROCES-DATA A/S: VIGO Users Manual for Version 3.0. - 1997.

ABBILDUNGSVERZEICHNIS

Abb. 1.1 Schichten des OSI-Referenzmodells ... 8
Abb. 1.2 Ein Überblick über die Internet-Protokolle ... 10
Abb. 2.1 Netzmanagmentsystem ... 13
Abb. 2.2 Komponenten eines verwalteten Knotens [ROS95] ... 14
Abb. 2.3 Das OSI-Modell für SNMP ... 16
Abb. 2.4 Der Baum für OBJECT IDENTIFIERS von Internet ... 22
Abb. 2.5 Die MIB-II Struktur ... 23
Abb. 2.6 Case-Diagramm [ROS93] ... 24
Abb. 2.7 Case-Diagramm für die `interface`-Gruppe [ROS93] ... 25
Abb. 2.8 Case-Diagramm für die `ip`-Gruppe [ROS93] ... 26
Abb. 2.9 Case-Diagramm für `tcp`-Gruppe [ROS93] ... 27
Abb. 2.10 Case-Diagramm für die `snmp`-Gruppe [ROS93] ... 27
Abb. 2.11 SNMP Nachricht in einer Übertragungsrahmen [MIL93] ... 30
Abb. 2.12 Struktur einer SNMP-Nachricht ... 30
Abb. 2.13 PDU-Struktur für Get-, GetNext-, SetRequest und GetResponse-Befehle ... 31
Abb. 2.14 PDU-Struktur für Trap-Meldungen ... 32
Abb. 2.15 Zusammenhang zwischen PDUs verschiedener SNMP-Versionen [STA98] ... 34
Abb. 3.1 P-NET mit verteilten Prozeßkomponenten [PNET96] ... 35
Abb. 3.2 Eine Multi-Net-Struktur mit P-NET [PNET96] ... 36
Abb. 3.3 Virtual Token Passing mit 3 Master. ... 38
Abb. 3.4 Einfache, komplexe und erweiterte Adresstypen ... 39
Abb. 3.5 Routing im P-NET ... 39
Abb. 3.6 Adressfeldumwandlung bei einer Anfrage ... 40
Abb. 3.7 Die Struktur des digitalen I/O-Channels ... 41
Abb. 3.8 Multitaskverfahren mit 4 Cyclic Tasks und 2 Softwire- bzw. Timedinterrupted Task ... 42
Abb. 3.9 Die Elemente von VIGO [VUM97] ... 43
Abb. 3.10 Virtuelles und physikalisches VIGO-Objekt [VUM97] ... 44
Abb. 3.11 MIB Editor [VUM97] ... 47
Abb. 3.12 Die Aufgabe eines IDCs [VUM97] ... 48
Abb. 4.1 OSI-Schichten einer Repeater-Kopplung ... 50
Abb. 4.2 OSI-Schichten einer .Bridge-Kopplung ... 50
Abb. 4.3 Router mit SNMP-Protokoll ... 51
Abb. 4.4 Router mit P-NET Protokoll ... 51
Abb. 4.5 Gateway zwischen LAN und P-NET ... 52
Abb. 4.6 Ein Netzmanagement-System für verschiedene Feldbusse ... 53
Abb. 4.7 Netzmanagement mit einem geteilten Agent ... 54
Abb. 4.8 Initialisierung des Sub-Agents ... 57
Abb. 4.9 Mögliche Speicherung der Netzwerk-Objekte im Sub-Agent ... 58
Abb. 4.10 Schrittweise Selektion der verwalteten Objekte [KKM97] ... 58
Abb. 4.11 Beispiel für Netzmanagmentsystem ... 60
Abb. 4.12 Protokollsequenz für das Lesen eines Wertes [KKM97] ... 63
Abb. 4.13 Initialisierungsphase zwischen Master- und Sub-Agent ... 65
Abb. 4.14 Kommunikation zwischen dem Master- und Sub-Agent nach der Initialisierungsphase ... 66
Abb. 5.1 Strukturogramm für die Initialisierungsphase des Konfigurators ... 71
Abb. 5.2 Die Benutzeroberfläche des Konfigurators ... 72
Abb. 5.3 Der "MIB View" Teil des Konfigurators ... 73
Abb. 5.4 Dialogfenster zur Anzeige der Objekteigenschaften ... 75

Abb. 6.1 Benutzeroberfläche des Sub-Agents 78
Abb. 6.2 Struktrogramm für die Initialisierungsphase des Sub-Agents 80
Abb. 6.3 Speicherung der SA-Objekte 82
Abb. 6.4 Multithreading im Sub-Agent 83
Abb. 6.5 Struktrogramm für den Verbindungsaufbau und Kommunikation 84
Abb. 6.6 Testprogramm für den Sub-Agent 87